Aviation Expert System

NIMALAN CHRISTOPHER
(WEK 990121)

Supervised by
Assoc. Prof. Dr. Roziati Zainuddin

Department of Artificial Intelligence

AVIATION EXPERT SYSTEM

NIMALAN CHRISTOPHER A/L ANANDA RAJAH

WEK990121

Supervised by

Assoc. Prof. Dr. Roziati Zainuddin

A Final Year Project, submitted as a partial requirement for the degree

BACHELOR OF COMPUTER SCIENCE

Faculty of Computer Science and Information Technology

UNIVERSITY MALAYA

Kuala Lumpur

2002

Abstract

The aviation expert system is a stand-alone rule-based prescription expert system. Prescription systems recommend solutions to a given system malfunction. The aviation expert system is an artificial intelligence system developed using a logic programming language. The primary objective of this system is to advise trainee pilots on the course of action to take when dealing with a failure situation in simulation cockpits. It receives input from the user and provides counter measures based on the input.

The problem-solving approach used in this system is rule-based reasoning. A rule is a form of procedural knowledge, which associates given information to some form of action. Reasoning is the process of working with knowledge, facts, and problem solving strategies to draw conclusions. Expert systems model the reasoning process of humans using a technique called inference, which is the process of deriving new information from known information.

The author hopes that the aviation expert system will overcome the disadvantages of the current system being employed by airline companies around the world.

Acknowledgements

The author would like to express his thanks to the following individuals for their support.

Assoc. Prof. Dr. Roziati Zainuddin, supervisor, for the guidance provided throughout the process of writing this thesis. The author is particularly grateful for the invaluable information and clarification provided.

Mr. Woo Chaw Seng, moderator, for a fair evaluation of this project, and positive criticisms and encouragement.

Mr. Arjuna Wijasuriya, First Officer, Malaysian Airlines System; domain expert, for providing data, information and expert advice on the aviation aspects of this study, as well as his time and effort.

If there are any flashes of brilliance in this document, these individuals deserve some of the credit. Any errors or omissions are solely the authors.

James, Prolog enthusiast, for his time and effort in sharing valuable knowledge on Visual Prolog.

Sharmila Rajah, freelance journalist, for the painstaking task of proofreading this document.

Last but not least, Tracy, PT, Thani, Raj, Vishnu, Hanu, and Alex Ross, who provided precious insight and encouragement the author needed to complete this thesis.

Nimalan Christopher,

23rd August 2002

Table of Contents

List of Tables

List of Figures

List of Abbreviations

1	Air Data Inertial Reference System	ADIRS
2	Air Data Inertial Reference Unit	ADIRU
3	Aircraft/ Airspace System	AAS
4	Revenue Passenger Miles	RPM
5	Artificial Intelligence	AI
6	Automatic Dependent Surveillance	ADS
7	Autopilot Flight Director Computer	AFDC
8	Autopilot Flight Director System	AFDS
9	Auxiliary Power Unit	APU
10	Control Display Unit	CDU
11	Distance Measuring Equipment	DME
12	Dynamic Link Library	DLL
13	Electrical Load Management System	ELMS
14	Electronic Engine Controller	EEC
15	Engine Indication and Crew Alerting System	EICAS
16	File Transfer Protocol	FTP
17	Flight Management System	FMS
18	Flight Management Computer	FMC
19	Flight Mode Control	FMC
20	Fourth Generation Language	4GL
21	Global Positioning System	GPS
22	Ground Proximity Warning System	GPWS
23	Graphical User Interface	GUI
24	High Frequency	HF
25	Human Computer Interaction	HCI
26	Hypertext Abstract Machine	HAM
27	Hyper Text Markup Language	HTML
28	Hyper Text Transfer Protocol	HTTP

29	Instrument Landing System	ILS
30	Integrated Drive Generators	IDG
31	Intelligent Aircraft/Airspace System	IAAS
32	Intelligent Rapid Prototyping System Selection	IRPS
33	Knowledge Specification Language	KSL
34	Logic Programming Associates	LPA
35	Mode Control Panel	MCP
36	Multifunction Display	MFD
37	Navigation Display	ND
38	Passenger Address	PA
39	Permanent Magnet Generators	PMG
40	Primary Flight Display	PFD
41	Prolog Development Center	PDC
42	Ram Air Turbine	RAT
43	Rapid Prototyping	RP
44	Satellite Communication	SATCOM
45	Secondary Attitude and Air Reference Unit	SAARU
46	Selective Calling	SELCAL
47	Terminal Collision Alerting System	TCAS
48	Traffic Management Agents	TrMA
49	Transmission Control Protocol/ Internet Protocol	TCP/IP
50	Transformer-Rectifier Unit	TRU
51	Very High Frequency	VHF
52	Visual Development Environment	VDE
53	Visual Programming Interface	VPI
54	World Wide Web	WWW

Chapter 1

Project Overview

1.0 Introduction

This chapter provides a short but clear description of what the final year project will consist of. The first four parts in this chapter cover the project definition, objective, scope and motivation. It explains the project, its goals, limitations and the reasons behind the approaches taken. The following part is the project timeline, which shows the start and end of each development phase of the project. The final part is the thesis organization, which is a brief description of each chapter in this report.

1.1 Project Definition

Commercial jet aviation is an exceptionally safe way to journey between places of great distances. Millions of people around the world fly safely on commercial aircrafts every day. In 1998, the world's commercial jet airlines carried approximately 1.3 billion people on 18 million flights while suffering only 10 fatal accidents [1]. It is very uncommon for a plane to crash because of a single contributing factor. One reason accidents are so rare is that commercial aviation has so many redundant, back-up systems to keep a problem from becoming serious [1]. Even though fatal jet accidents are rare, the aviation community worldwide is continuing to work together to reduce them.

The flight crew is provided with 'checklists' that are used from the time the aircraft is parked at the gate, before and after takeoff, and until the aircraft lands and parks at the gate at the next airport. Checklists contain, in abbreviated form, information required by the trained flight crew to operate the airplane in normal situations and to cope with non-normal situations [2].

There are two different checklists, which are the normal checklist and the non-normal checklist. Normal checklists are organized by phase of flight (preflight, before start, after start, before takeoff, after takeoff, approach, landing, parking, secure) and are used to verify that certain critical procedural steps have been accomplished [2]. However, only important procedural steps that are vital to normal operations are included in this checklist.

The Engine Indication and Crew Alerting System (EICAS), which displays alert messages that indicates a failure condition, calls for the non-normal checklists to be selected and accomplished. The checklist titles are the same as the alert messages. Checklists that do not correspond to EICAS alert messages are called un-annunciated checklists. There are 16 sections in the non-normal checklist, 14 of which are systems. They are:

- Un-annunciated Checklists
- Airplane General, Emergency Equipment, Doors, Windows
- Air Systems
- Anti-Ice, Rain
- Automatic Flight
- Communications
- Electrical
- Engines, Auxiliary Power Unit (APU)
- Fire Protection
- Flight Controls
- Flight Instruments, Displays
- Flight Management, Navigation
- Fuel
- Hydraulics
- Landing Gear
- Warning Systems

While every attempt is made to establish necessary non-normal checklists, it is not possible to develop checklists for all conceivable situations, especially those involving multiple failures. In certain unrelated multiple failure situations, the flight crew may have to combine elements of more than one checklist and/or exercise judgment to determine the safest course of action [2]. In this study, the author has outlined a system that will be able to help trainee pilots in simulation cockpits and at the same time overcome the weakness of the non-normal checklist of not being able to cope with multiple failures.

1.2 Project Objective

Items on a checklist are categorized as recall items or reference items. Recall items are critical steps that must be accomplished from memory while reference items are actions to be accomplished while reading the checklist. This is why the author proposes that this system be used for trainee pilots in simulation cockpits. Since a trained pilot must already know all recall items by heart, it is only wise to use this system as a basis for training a pilot.

In order for this system to be more accessible to trainee pilots, the author has decided that the information in the checklist will not be in abbreviated form. This is because trained pilots are already accustomed to the information in the checklist, making it easier for them to read the checklist compared to a trainee pilot.

As previously mentioned, in certain unrelated multiple failure situations, the flight crew may have to combine elements of more than one checklist and/or exercise judgment to determine the safest course of action. Checklists cannot be created for all conceivable situations and are not intended to replace good judgment. However, the author has decided to include solutions for multiple failure situations in this system, which will be gained from a series of interviews with the domain expert. It is important to note that the solutions provided are based on experience and good judgment and therefore will differ from one domain expert to another. These solutions will give the trainee pilots an advantage in dealing with multiple failure situations, something that usually comes with experience.

Basically, the system will be able to fulfill these objectives:

- Advise trainee pilots on counter measures to take if there is an alert message on the EICAS screen in the simulation cockpit.
- Advise trainee pilots on counter measures to take if there are multiple alert messages on the EICAS screen in the simulation cockpit.
- Give trainee pilots faster and easier access to the checklists.

1.3 Project Scope

This project aims to implement a Rule-Based Expert System that advises trainee pilots on what counter measures to take when an alert message is displayed. A rule's structure logically connects one or more antecedents (also called premises) contained in the IF part, to one or more consequents (also called conclusions) contained in the THEN part [3].

Items in the normal checklist are critical procedural steps that have to be taken at different phases of a flight. The flight crew uses normal checklists after accomplishing all applicable procedural items [2]. This means that the actions have already been taken and the checklist only serves the purpose of verifying whether these steps have been accomplished. There aren't pairs of premises and conclusions to form rules. Therefore, the inclusion of the normal checklist in the system is only for knowledge purposes.

In the non-normal checklist, four out of the 16 sections will be included in the system only for knowledge purposes because of similar circumstances. These four sections are:

- Automatic Flight
- Communications
- Flight Instruments, Displays
- Warning Systems

The items on the checklists for these sections are just conditions which require no counter measures. They simply indicate the status of certain equipments in a particular system. Therefore, only 12 sections in the non-normal checklist will be able to recommend counter measures when alert messages are displayed. They are:

- Un-annunciated Checklists
- Airplane General, Emergency Equipment, Doors, Windows
- Air Systems
- Anti-Ice, Rain
- Electrical

- Engines, Auxiliary Power Unit (APU)
- Fire Protection
- Flight Controls
- Flight Management, Navigation
- Fuel
- Hydraulics
- Landing Gear

The technical descriptions given in this project are for the Boeing 777 and may differ a little from other commercial airplanes.

The Visual Prolog language will be used for the development of the system. Reasons for this will be illustrated in more detail in a later chapter.

1.4 Motivation

Rule-based systems are the most popular choice of knowledge engineers for building an expert system. This popularity is due to the large number of successful rule-based systems built and the considerable amount of development software available. Expert systems have a major advantage over all the learning systems in that real data is not required in order to develop the system. However, a knowledgeable expert is required.

The many advantages of rule-based systems are another reason the author has chosen this form of approach. Rules can be used to represent heuristics or 'rules of thumb', which are 'tricks of the trade' that the domain expert may have learnt from experience and are often more valuable than fundamental principles learnt from books. Rule-based systems also allow statements with uncertainty to be captured as rules. The separation of knowledge in the knowledge base, from its control performed by the inference engine, allows the change of the system's knowledge or control separately and also permits the addition of rules anywhere in the knowledge base. Besides that, rule-based systems have a proportional growth of intelligence, that is, as the number of rules increase, the system's level of intelligence about the problem also increases.

The implementation of this system would benefit trainee pilots in many ways. They would have a head start in dealing with multiple failures during a flight, an advantage over their predecessors, who have acquired the knowledge through years of experience.

1.5 Project Timeline

The Gantt chart below shows the start and end of each phase of the project and the time in which this project is expected to reach completion.

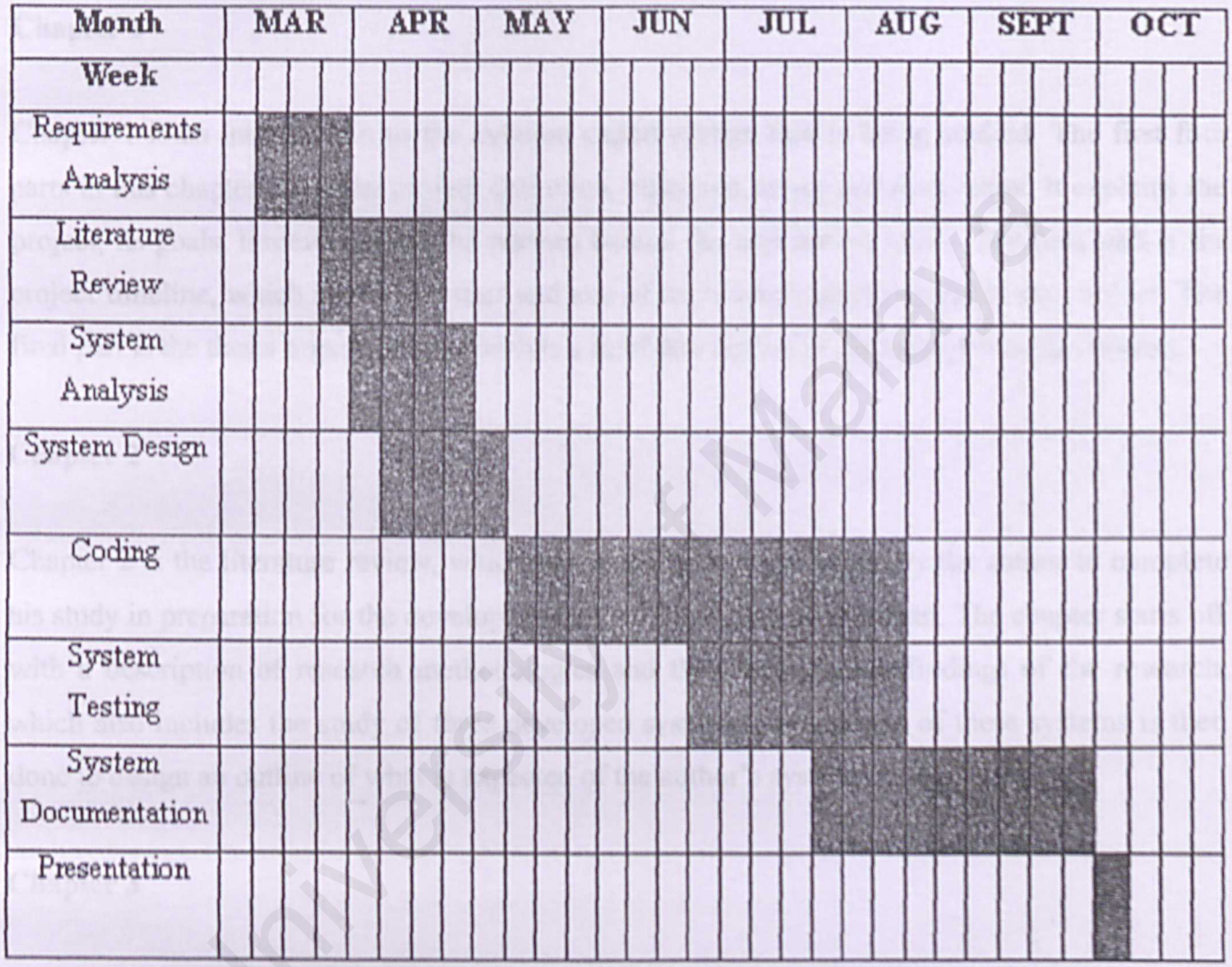


Figure 1.0 Project Timeline

1.6 Thesis Organization

This thesis is divided into five parts. Below is a list of chapters in this book that fulfills the requirement of this thesis, along with a brief explanation of what its contents are about.

Chapter 1

Chapter 1 is an introduction to the aviation expert system that is being studied. The first four parts in this chapter cover the project definition, objective, scope and motivation. It explains the project, its goals, limitations and the reasons behind the approaches taken. The next part is the project timeline, which shows the start and end of each development phase of the project. The final part is the thesis organization, which is a brief description of each chapter in this project.

Chapter 2

Chapter 2 is the literature review, which covers all the research done by the author to complete his study in preparation for the development of the aviation expert system. The chapter starts off with a description of research methodologies and then lays out the findings of the research, which also includes the study of three developed systems. An analysis of these systems is then done to design an outline of what is expected of the author's system.

Chapter 3

Chapter 3 covers the methodology used to develop the system. It describes the type of methodology used and the project schedule. It also describes rule-based reasoning, which is a requirement of the system. How rules work and other relevant details pertaining to rules are discussed in further detail. The structure and characteristics of expert systems are also included.

Chapter 4

Chapter 4 is an analysis on the requirements of the system, which is made up of functional and non-functional requirements. Software and hardware requirements are also included for system development purposes. This chapter also describes the system design and goes into detail about the architecture of the system.

Chapter 5

Chapter 5 elaborates on the implementation of the Aviation Expert System, which includes software and hardware used in accomplishing this project.

Chapter 6

This chapter describes the testing phase of the system, which comprises of Unit Testing, Integration Testing and System Testing. Detailed explanation of these phases is discussed thoroughly here.

Chapter 7

The evaluation phase of the system, which includes system strengths and limitations, is documented in this chapter. Problems faced by the author and solutions to it, are also included. Future enhancements that could be implemented are discussed in detail.

Chapter 8

Chapter 8 is the conclusion of the thesis. It summarizes what has been done in this project, and discusses suggestions to improve the system.

1.7 Conclusion

As of this point, we have a clearer picture of what the aviation expert system is about and how it is aimed at training pilots in dealing with failures in simulation cockpits. Before getting started on the system development phase, it is important to gain a clearer insight on certain areas of knowledge pertaining to the system. This task will be carried out in the next chapter, the literature review.

Chapter 2

Literature Review

2.0 Introduction

A literature review is a classification of what accredited scholars or researches have written on a topic, organized according to a guiding concept such as your research objective, thesis, or the issue you wish to address [4]. In this chapter the author lays out the research he has done. There are three systems the author wishes to use as a reference in developing the aviation expert system. Based on the research done, the author performs an analysis and derives an appropriate synthesis of his system.

2.1 Research Methodology

There are several materials and methods that the author used in carrying out his research. They are:

- Internet
- Books
- Manuals
- Interviews
- Final Year Projects

2.1.1 Internet

The Internet is a public, cooperative, and self-sustaining facility accessible to hundreds of millions of people worldwide. Physically, the Internet uses a portion of the total resources of the currently existing public telecommunication networks. Technically, what distinguishes the Internet is its use of a set of protocols called TCP/IP (Transmission Control Protocol/Internet Protocol). The most widely used part of the Internet is the World Wide Web, which gives us access to millions of pages of information.

The author used several search engines that proved to be very useful in providing information that is not available or hard to find in print. The search engines used were:

- Google
- Yahoo
- Ask
- Lycos
- Metacrawler
- AltaVista

2.1.2 Books

Other than the Internet, the author also used books, which were obtained from the library and other personal sources, as a means of finding out information. Books that have been used in other courses as part of the author's degree program also proved to be very useful. All the books that were used are listed in the 'Reference' section.

2.1.3 Manuals

The domain expert provided the author with operations manuals, which contributed significantly to the author's research on the aviation aspect of this project. The normal and non-normal checklists (described in Chapter 1) are also available in these manuals. All the manuals that were used are listed in the 'Reference' section.

2.1.4 Interviews

Direct elicitation methods involve the articulation by the domain expert of the problem-solving knowledge and primarily include interviews and case studies [3]. Interviews are the most common knowledge elicitation technique used in the design of expert systems. The domain expert was very cooperative in sharing his knowledge and conveyed the technical aspects in a very clear and understandable manner. The questions used for the interview can be found in 'Appendix B'.

2.1.5 Final Year Projects

The Faculty of Computer Science and Information Technology degree students' final year projects were used as a reference, especially when the author had doubts about the format of the report. These final year projects are stored in the Document Room in the faculty.

2.2 Findings

The first part of this section provides an insight on aviation. It gives a brief description on aircraft parts and their functions and also touches on flight dynamics. This is essential in achieving a better understanding of the functions of the system. The next part is an introduction to expert systems, the need for them and how they are used. The author then studies three developed systems in the hopes of gaining a clearer picture on how to develop his system. An analysis of these systems is then done to come up with an outline of what is expected of the author's system.

2.2.1 Aviation

2.2.1.1 Main Components Of An Airplane [5]

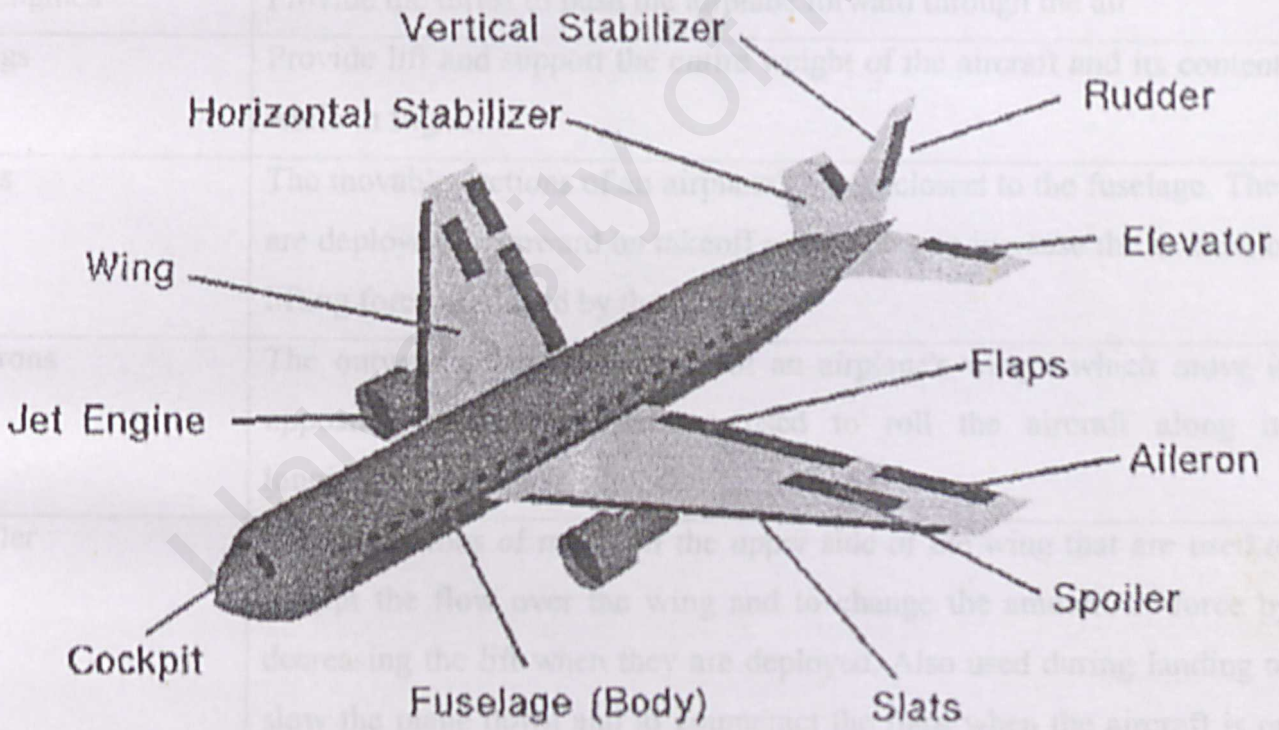


Figure 2.0 Main Components of an Airplane

An airplane is a vehicle heavier than air, powered by one or more engines, which travels through the air by reaction of air passing over its wings. Figure 2.0 shows the main components of an airplane. These components, which are the moveable outer surfaces of an airplane, are better known as flight control surfaces. These surfaces control the flow of air over the various sections of the aircraft causing it to move in different ways. Inside the cockpit, pilots control the movement of the surfaces with their hands or feet by pushing, pulling or turning the controls to make the airplane move in the proper manner. The table below lists the main components of an airplane and their functions.

Table 2.0 Airplane Components and Their Functions

Components	**Function**
Fuselage	The central body portion of an airplane, which accommodates the crew and passengers or cargo.
Cockpit	The space or compartment from which the airplane is piloted.
Jet Engines	Provide the thrust to push the airplane forward through the air.
Wings	Provide lift and support the entire weight of the aircraft and its contents while in flight.
Flaps	The movable sections of an airplane's wings closest to the fuselage. They are deployed downward on takeoff and landing to increase the amount of lifting force produced by the wing.
Ailerons	The outward movable sections of an airplane's wings, which move in opposite directions. They are used to roll the aircraft along its longitudinal axis.
Spoiler	Square sections of metal on the upper side of the wing that are used to disrupt the flow over the wing and to change the amount of force by decreasing the lift when they are deployed. Also used during landing to slow the plane down and to counteract the flaps when the aircraft is on the ground.
Slats	Extended on takeoff and landing to increase the amount of lifting force produced by the wing.
Vertical Stabilizer	Keeps the nose of the plane from swinging along its vertical axis.

Horizontal Stabilizer	Prevents an up-and-down motion of the nose along its lateral axis.
Rudder	The hinged part of the vertical stabilizer, which is used to deflect the tail to the left and right.
Elevator	The hinged part of the horizontal stabilizer, which is used to deflect the tail up and down.

2.2.1.2 What Makes An Airplane Fly? [5]

There are four forces acting on an airplane in flight, which are thrust, drag, lift and weight (gravity). Thrust is the force exerted by the engine and its propellers, which pushes air backward with the object of causing a reaction of the airplane in the forward direction. Drag is the resistance of the airplane to forward motion directly opposed to thrust. Lift is the upward force created by the wings moving through the air, which sustains the airplane in flight. Weight is the downward force due to the weight (gravity) of the airplane and its load, directly opposed to lift.

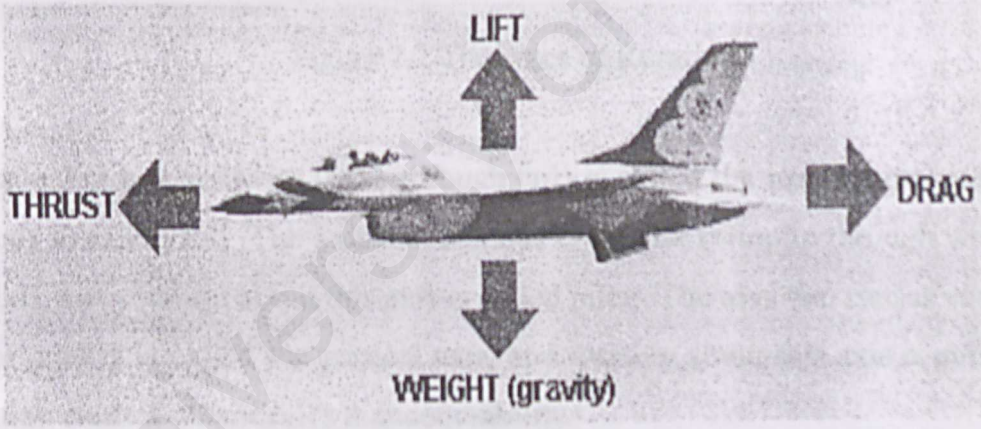


Figure 2.1 Forces Acting on an Airplane in Flight

When thrust and drag are equal and opposite, the airplane is said to be in a state of equilibrium. This means it will continue to move forward at the same uniform speed. If either of these forces becomes greater than the force opposing it, the state of equilibrium will be lost. If thrust is greater than drag, the airplane will accelerate or gain speed. If drag is greater than thrust, the airplane will decelerate or lose speed and will descend. Similarly, when lift and weight are equal and opposite, the airplane will be in equilibrium. If lift, however, is greater than weight, the airplane will climb. If weight is greater than lift, the airplane will sink.

Aircraft fly in three dimensions, and they move in directions other than straight and level. In addition to moving forward, an aircraft in flight may move about three axes.

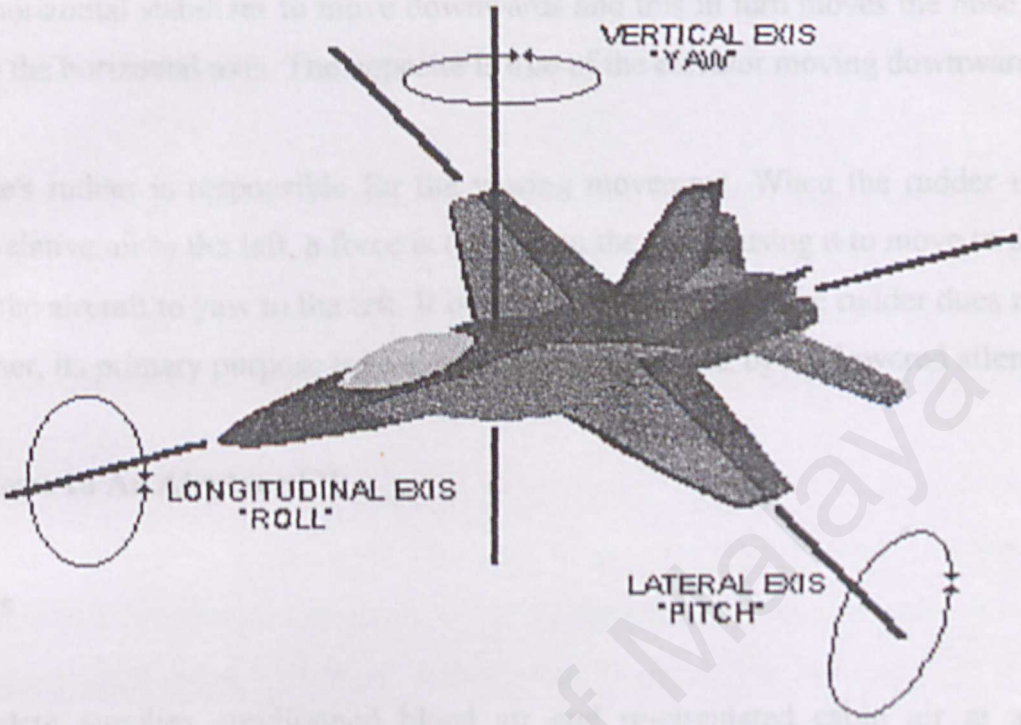


Figure 2.2 The Axes of Rotation

The axis that extends lengthwise (nose through tail) is called the longitudinal axis, and rotation about this axis is called roll. The axis that extends crosswise (wingtip through wingtip) is called the lateral axis, and rotation about this axis is called pitch. The axis that passes vertically through the centre of gravity is called the vertical axis, and rotation about this axis is called yaw. These axes of rotation control an aircraft's manoeuvrability.

Ailerons cause an airplane to roll about the longitudinal axis. When an aileron is not in perfect alignment with the total wing, it changes the wing's lift characteristics. To make a wing move upward, the aileron on that wing must move downward. When this happens, the total lift being produced by that wing is increased. At the same time, the lift on the other wing is reduced. This causes the aircraft to roll.

The elevator, which is attached to the horizontal stabilizer, is what that causes the pitching movement. When the elevator is moved upward, the relative curvature of the under side of the horizontal stabilizer is increased thus increasing the lifting force on the under side of it. This causes the horizontal stabilizer to move downwards and this in turn moves the nose upwards in reference to the horizontal axis. The opposite is true of the elevator moving downwards.

The airplane's rudder is responsible for the yawing movement. When the rudder is moved to deflect the relative air to the left, a force is created on the tail, causing it to move to the right and the nose of the aircraft to yaw to the left. It is important to note that the rudder does not steer the aircraft; rather, its primary purpose is to offset the drag produced by the lowered aileron.

2.2.1.3 Systems In An Airplane [2]

Air Systems

The air system supplies conditioned bleed air and re-circulated cabin air at a controlled temperature throughout the airplane. The system supplies conditioned air to the flight deck shoulder heaters. It also supplies ventilation for the passenger cabin (lavatories, galleys, individual passenger seat gaspers), flight deck crew rest compartment and lower crew rest compartment.

Two identical air conditioning packs cool bleed air from the engines, APU, or high-pressure air from a ground source. Bleed air is pre-cooled before entering the pack. Two identical pack controllers control the packs. If a controller fails, pack control switches automatically to the other controller. Each pack can operate at reduced flow during certain phases of flight to reduce fuel consumption.

Anti-Ice, Rain

The anti-ice and rain systems include:

- Automatic ice detection

- Engine anti-ice
- Wing anti-ice
- Flight deck window heat
- Windshield wipers
- Probe heat

The automatic ice detection system detects airplane icing in flight. Automatic ice detection can be operated in flight only. The system provides signals to control the engine and wing anti-ice systems when those systems are in automatic mode. The system consists of two ice detector probes, one on each side of the forward fuselage. If the probes detect ice build-up, they are automatically de-iced.

The engine anti-ice system uses engine bleed air to provide engine cowl inlet ice protection. Engine anti-ice can be operated in flight or on the ground. The engine anti-ice annunciation appears on the EICAS screen when an engine valve is open. The left and right engines have identical, independent anti-ice system. This allows the remaining system to operate if one engine fails.

The wing anti-ice system provides bleed air to three mid-wing leading edge slats on each wing. Wing-anti ice can be operated in flight only. If the total air temperature is 10 C^{o} or above, both manual and automatic wing anti-ice operation is inhibited for five minutes after takeoff. It is made available again when the total air temperature drops below 10 C^{o}. The wing anti-ice annunciation is displayed on the EICAS screen when a wing anti-ice valve is open. If a bleed source is lost and bleed duct isolation has not occurred, the isolation valves automatically open to maintain anti-icing to both wings.

All flight deck windows are electrically heated. The forward windows have exterior surface anti-icing, and exterior and interior surface antifogging protection. The side windows have interior surface antifogging protection only. Window heat operates as soon as electrical power is established. The windows are protected from thermal shock. A backup antifogging system for the forward windows operates automatically if there is a primary window heat system failure.

The rain removal system for the forward windows consists of wipers and a permanent rain repellent coating on the windows. The forward windows are equipped with independently controlled, two-speed wipers.

There are three heated probes, one on the left and two on the right side of the forward fuselage. On the ground with either engine operating, the probes are automatically heated at reduced power to avoid overheating. In flight, they are automatically heated at full power.

Automatic Flight

The automatic flight control system consists of the Autopilot Flight Director System (AFDS) and the auto throttle system. Both these systems are controlled using the Mode Control Panel (MCP) and the Flight Management Computers (FMC). Normally, the AFDS and auto throttle are controlled automatically by the flight mode control to perform climb, cruise, descent, and approach flight path guidance.

The AFDS consists of three Autopilot Flight Director Computers (AFDC) and the MCP. The MCP provides an interface for the pilot to control the autopilot, flight director, altitude alert, and auto throttle systems. The MCP is used to select and activate AFDS modes, and establish altitudes, speeds, and climb/descent profiles. The AFDC provides control of the flight directors, and autopilot. Flight director information is displayed on the primary flight displays. The AFDS does not have direct control of the flight control surfaces. The autopilot controls the elevators, ailerons, flaps, and spoilers through the fly-by-wire flight control system. Autopilot rudder commands are added only during an autopilot approach and landing. The autopilot controls nose wheel steering during rollout after an automatic landing.

The auto throttle system provides automatic thrust control from takeoff through landing. Auto throttle operation is controlled from the MCP and the Control Display Units (CDU). The MCP provides mode and speed selection. The CDU provides FMC thrust reference mode selection. The auto throttle can be operated without using the flight director or the autopilot. When the auto throttle is used during a manual landing, thrust reduces at 25 feet radio altitude. The auto throttle can be manually overridden or disconnected by pushing either auto throttle disconnect switches.

Communications

The communication systems include:

- Cockpit voice recorder
- Radio communication control
- Selective Calling (SELCAL) system
- Satellite Communication (SATCOM) system
- Communication crew alerting system
- Interphone communication system
- Data communication system

The communication systems are controlled using the:

- Audio control panels
- Radio tuning panels
- Control Display Unit (CDU) communications pages
- Multifunction Display (MFD) communication pages

The cockpit voice recorder records any transmissions from the fight deck made through the audio control panels. It also records any transmissions from the flight deck area conversations using an area microphone. All inputs are recorded continuously in a 30-minute loop tape recorder.

The radio communication systems consist of the Very High Frequency (VHF) communication system, the High Frequency (HF) communication system, the SELCAL system and the SATCOM system.

Three independent VHF voice/data radios, designated VHF Left, VHF Centre, and VHF Right are installed. Any VHF radio can be controlled by any radio tuning panel. The audio control panels are used to control voice transmission and receiver monitoring. VHF Left is configured for voice communication only. VHF Centre and VHF Right can be configured for data or voice communication. However, only one VHF radio can operate in the data mode at a time. Data communication is normally selected on VHF Centre.

There are two independent HF communication radios, designated HF Left and HF Right. Each HF radio can be tuned from any radio tuning panel. HF radio sensitivity can only be set on the on-side radio tuning panel. The audio tuning panels are used to control voice transmission and receiver monitoring. When an HF transmitter is keyed after a frequency change, the antenna tunes. While the antenna is being tuned, a tone can be heard through the audio system. Both HF radios use a common antenna. When either HF radio is transmitting, the antenna is disconnected form the other HF radio, and it cannot be used to transmit or receive. However, both HF radios can receive simultaneously if neither is being used for transmission.

The SELCAL system monitors the three VHF radios and the two HF radios. When the system receives a call from a ground station, the crew is alerted through the communication crew alerting system.

The SATCOM system provides both data and voice communications. The satellite data unit is controlled through the CDU. Voice transmission is controlled using CDU and the audio control panel. Calls can be initiated using the CDU. Directories of airline-defined numbers are line selectable or manual numbers can be entered. The SATCOM CDU control pages are displayed by selecting SAT on the MENU page.

The communication crew alerting system provides aural and visual alerts for normal operations requiring crew awareness that may require crew action. Visual alerts are presented as EICAS messages preceded by a bullet symbol (•). The aural alert is a high-low chime.

The interphone communication system includes the:

- Flight interphone system
- Cabin interphone system
- Service interphone system
- Passenger Address (PA) system

The flight interphone system provides communications on the flight deck and between the flight deck and the ground crew through the flight interphone jack on the APU ground control fire protection panel in the nose landing gear wheel well.

The service interphone system provides voice communications between ground crew stations at various locations around the airplane. The system can be connected to the flight interphone system through the service interphone switch on the overhead panel.

The cabin interphone system provides voice communications between flight deck and the flight attendant stations. Boom microphones, oxygen mask microphones, and hand microphones are selected and used for communication. A cabin interphone station(s) must be selected and a call initiated from the centre CDU to alert the desired station to pick up the call.

The flight crew uses the Passenger Address (PA) system to make cabin announcements. Pushing a PA transmitter select switch on an audio control panel and activation of a microphone switch provides direct access to all PA areas.

The audio control panels are used to manage the radio and interphone communications systems. Navigation receiver audio can also be monitored. The captain, first officer, and first observer audio control panels are installed on the aft isle stand. Systems are monitored using headphones or speakers. An oxygen mask microphone is enabled and the boom microphone is disabled when the oxygen mask stowage doors are open. The oxygen mask microphone is disabled and the boom microphone is enabled when the left oxygen mask stowage box door is closed and the RESET/TEST lever is pushed.

The radio tuning panels are used to tune the VHF and HF radios. The panels are designated left, centre, and right and are normally associated wit the respective VHF and HF radios.

Electrical

The electrical system generates and distributes AC and DC power to other airplane systems. System operation is automatic. Electrical faults and automatically detected and isolated. The electrical system is comprised of:

- Main AC power
- Backup power
- DC power
- Standby power
- Flight controls power

The AC electrical system is the main source for airplane electrical power. The Electrical Load Management System (ELMS) provides load management and protection to ensure power is available to critical and essential equipment. If the electrical loads exceed the power available, ELMS automatically sheds AC loads by priority until the loads are within the capacity of the airplane or ground power generators. When an additional power source becomes available or the load decreases, ELMS restores power to shed systems. The entire airplane AC electrical load can be supplied by any two main AC power sources. The main AC electrical sources are:

- Left and right engine Integrated Drive Generators (IDG)
- APU generator
- Primary and secondary external power

The backup electrical system is designed to automatically provide power to selected airplane systems. The backup electrical system automatically powers one or both transfer busses when:

- Only one main AC generator is available
- Power to one or both of the main AC busses is lost
- The system is automatically tested after engine starts

Backup power is provided by one variable speed, variable frequency generator mounted on each engine. A frequency converter converts the generator frequency to a constant 400 Hz. Only one backup generator can power the converter at a time. Each backup generator contains two

Permanent Magnet Generators (PMGs) that supply power to the flight control DC electrical system.

The DC electrical system includes the main DC electrical system and the flight control DC electrical system. The main DC electrical system uses Transformer-Rectifier Units (TRU) to produce DC power. AC transfer busses power the TRUs. The Centre No.1 TRU powers the captain's flight instrument bus and the battery bus. The captain's flight instrument bus provides a second DC power source for the right flight control bus and the left main DC bus. The Centre No.2 TRU powers the first officer's flight instruments bus. The first officer's flight instruments bus provides a second DC powers source for the captain's instruments bus.

The flight control DC electrical system is a dedicated power system that supplies DC power to the primary flight control system. The primary electrical power sources for the flight control system are left and right PMGs, housed within the backup generators. If the PMGs are not available, the left and right flight control busses are powered by the main DC busses. The centre flight control bus is powered by different power sources through the captain's flight instrument bus. The hot battery bus provides another backup power source for the left and centre flight control busses only. Individual batteries are connected to each flight control DC bus to provide continuous flight control DC power during source transfers. These batteries are capable of supplying power for only a short period of time.

The standby electrical system can supply DC and AC power to selected flight instruments, communications and navigation systems, and the flight control system, if there are primary AC and DC electrical power system failures.

Engines, APU

The airplane is powered by two Rolls Royce Trent 892 engines. The engines are rated at 90, 000 pounds of takeoff thrust each. The engines are three-rotor axial flow turbofans of high compression and bypass ratio. The N1 rotor consists of the fan and a low-pressure turbine section on a common shaft. The N2 rotor consists of an intermediate pressure compressor section and an

intermediate pressure turbine section on a common shaft. The N3 rotor consists of a high-pressure compressor section and a high-pressure turbine section on common shaft. The N1, N2, and N3 rotors are mechanically independent. The N3 rotor drives the engine accessory gearbox. Each engine is controlled by an Electronic Engine Controller (EEC). The EEC monitors auto throttle and flight crew inputs through the thrust lever levers to automatically control the engines. Each engine has individual flight deck controls. Thrust is set by the positioning of the thrust levers. The thrust levers are positioned automatically by the auto throttle system or manually by the flight crew.

The Auxiliary Power Unit (APU) is a self-contained gas turbine engine located in the airplane tail cone. The APU can be started and operated to the airplane maximum certified altitude. The APU supplies bleed air and electrical power. Electrical power has priority over bleed air. Electrical power is available throughout the airplane operating envelope. Bleed air is available at or below 20,000 ft.

Fire Protection

There are fire protection detection and extinguishing systems for the:

- APU
- Cargo compartments
- Engines
- Lavatories

The flight deck crew rest compartment has a fire detection system, but no fire extinguishing system. The lower crew rest compartment centre has a fire detection system and a manually activated fire extinguishing system. The passenger business communication centre has a smoke detection system, but no fire extinguishing system. The engines also have overheat detection system. The main gear wheel wells have a fire detection system, but no fire extinguishing system.

Flight Controls

The primary control system uses conventional control wheel, column, and pedal inputs from the pilot to electronically command the flight control surfaces. The system provides conventional control feel and pitch responses to speed and trim changes. The system electronic components provide enhanced handling qualities to reduce pilot workload.

The primary flight control system is highly redundant, with three operating modes: normal mode, secondary mode, and direct mode. The primary flight controls are powered by redundant hydraulic sources. The secondary flight controls, high lift devices consisting of flaps and slats, are hydraulically powered with an electronic powered backup system.

The pilot controls consists of:

- Two control columns
- Two control wheels
- Two pairs of rudder pedals
- Control wheel pitch trim switches
- Alternate pitch trim levers
- The speed brake lever
- The flap lever
- Aileron trim switches
- Rudder trim selector
- Manual rudder trim cancel switch

Flight Instruments, Displays

The flight instruments and displays supply information to the flight crew on six flat panel crystal display units. The units display for primary groups of information:

- The Primary Flight Display (PFD)
- The Navigation Display (ND)
- The Engine Indication and Crew Alerting System (EICAS)

- The Multifunction Display (MFD)

The PFD presents a dynamic colour display of all the parameters necessary for flight path control. The PFD provides the following information:

- Flight mode annunciation
- Airspeed
- Altitude
- Vertical speed
- Attitude
- Steering information
- Radio altitude
- Instrument landing display
- Approach minimums
- Heading/track indications
- Engine fail

The ND provides a mode-selectable colour flight progress display. The modes are:

- MAP
- VOR
- APP (approach)
- PLN (plan)

The MAP mode is recommended for most phases of flight. This mode shows airplane position relative to the route of flight against a moving map background. The VOR and APP modes display track, heading, and wind speed and direction while the PLN mode displays the active route.

Flight Management, Navigation

Navigation systems include Global Positioning System (GPS), Air Data Inertial Reference System (ADIRS), Distance Measuring Equipment (DME), Instrument Landing System (ILS), weather radar and the Flight Management System (FMS).

The left and Right GPS receivers are independent and supply very accurate geographical data to the FMC. All GPS tuning is automatic.

The ADIRS calculates airplane position, speed, altitude, and attitude data for the displays, FMS, flight controls, engine controls, and other systems. The major components of ADIRS are the Air Data Inertial Reference Unit (ADIRU), Secondary Attitude and Air Reference Unit (SAARU), and air data modules. The ADIRU supplies primary fight data, inertial reference, and air data. The ADIRU is fault-tolerant and fully redundant. The SAARU is a secondary source of critical flight data for displays, flight control systems, and other systems. If the ADIRU fails, the SAARU automatically supplies attitude, heading, and air data.

Two DME systems are installed. The DMEs are usually tuned by the Flight Management Computer (FMC), but may be tuned manually. Three ILS receivers are installed. They are usually tuned by the FMC, but can also be tuned manually. The weather radar system consists of two receiver-transmitter units, an antenna, and a control panel. Radar returns display on the Navigation Display (ND). Turbulence can be sensed by the weather radar only when there is sufficient precipitation. Clear air turbulence cannot be sensed by radar.

The FMS aids the flight crew with navigation, in-flight performance optimization, automatic fuel monitoring, and flight deck displays. Automatic flight functions manage the airplane lateral flight path and vertical flight path. The displays include a map for airplane orientation and command markers on the airspeed, altitude, and thrust indicators to help in flying efficient profiles. The flight crew enters the applicable route and flight data into the CDUs. The FMS then uses the navigation database, airplane position, and supporting system data to calculate commands for manual and automatic flight path control. The FMS tunes the navigation radios

and sets courses. The FMS navigation database supplies the necessary data to fly routes, holding patterns, and procedure turns. Cruise altitudes and crossing altitude restrictions are used to calculate vertical flight path commands. Lateral offsets from the programmed route can be calculated and commanded.

Fuel

The fuel system supplies fuel to the engines and the APU. The fuel is contained in a centre tank, and left and right main tanks.

Fuel quantity is measured by sensors in each tank. Total fuel quantity is displayed on the primary EICAS display. Tank quantities and total fuel quantity are displayed on the FUEL synoptic display. Expanded fuel indications showing the left main, centre, and right main tank quantities are displayed when non-normal conditions occur.

Fuel temperature is displayed on the primary EICAS display. The temperature is normally displayed in white. It is displayed in amber when the fuel temperature approaches the fuel freeze temperature entered on the FMS CDU. During jettison, the TO REMAIN quantity replaces the EICAS display fuel temperature indication. Fuel temperature and minimum fuel temperature are also displayed on the fuel synoptic display.

Each fuel tank contains two AC-powered fuel pumps. A single pump can supply sufficient fuel to operate one engine under all conditions. The two centre tank fuel pumps are override/jettison pumps. These pumps have a higher output pressure than the left and right main tank fuel pumps. The centre tank pumps override the main tank pumps so that the centre tank fuel is used before wing tank fuel.

The fuel manifolds are arranged so that any fuel tank pump can supply either engine. The crossfeed valves are closed during normal operations. The closed crossfeed valves isolate the left and right system. Either valve can be opened to feed an engine from the opposite fuel tank.

When the fuel quantities in the left and right main tanks differ by a specified quantity, the EICAS alert message FUEL IMBALANCE displays. Fuel balancing is accomplished by opening one or both crossfeed valves and turning off the fuel pump switches for the fuel tank that has the lower quantity. Fuel balancing may be done in any phase of flight.

APU fuel is supplied from the left fuel manifold. APU fuel can be provided by any AC fuel pump supplying fuel to the left fuel manifold or by the left main tank DC fuel pump.

The fuel jettison system allows jettison from all fuel tanks. Fuel is jettisoned through jettison nozzle valves inboard of each aileron. Jettison pumps in the main tanks and override/jettison pumps in the centre tank pump fuel overboard through jettison nozzles valves.

Hydraulics

The airplane has three independent hydraulic systems: left, right, and centre. The hydraulics systems power the:

- Flight controls
- Leading edge slats
- Trailing edge flaps
- Landing gear
- Wheel brakes
- Nose and main gear steering
- Thrust reversers

Flight control system components are distributed so that any one hydraulic system can provide adequate airplane controllability. Hydraulic fluid is supplied to each hydraulic pump from the associated system reservoir. The reservoirs are pressurized by the bleed air system.
The left and right hydraulic systems are identical. They differ only in the components that they power. The left hydraulic system powers:

- Flight controls
- The left engine thrust reverser

The right hydraulic system powers:

- Flight controls
- Normal brakes
- The right engine thrust reverser

The centre hydraulics system powers:

- Flight controls
- Leading edge slats
- Trailing edge flaps
- Landing gear actuation
- Alternate brakes
- Reserve brakes
- Nose gear steering
- Main gear steering

The Ram Air Turbine (RAT), when deployed, provides hydraulic power only to the primary flight control components connected to the centre hydraulic system. The RAT provides hydraulic and electrical power throughout the flight envelope. In flight, the RAT deploys automatically if:

- Both engines are failed and centre system pressure is low, or
- Both AC transfer busses are not powered, or
- All three hydraulic system pressures are low.

Landing Gear

The airplane has two main landing gears and a single nose gear. The nose gear is a conventional steerable two-wheel unit. Each main gear has six wheels in tandem pairs. To improve turning radius, the aft axle of each main gear is steerable. Hydraulic power for retraction, extension, and steering is supplied by the centre hydraulic system. An alternate extension system is also provided. The normal brake hydraulic system is powered by the right hydraulic system. The alternate/reserve brake hydraulic system is powered by the centre hydraulic system. Antiskid protection is provided with both systems, but the autobrake system is available only through the

normal system. A brake temperature monitor system and tire pressure indication system displays each brake temperature and tire pressure on the GEAR synoptic display.

In-flight and ground operations of various airplane systems are controlled by the air/ground sensing system. The system receives air/ground logic signals from sensors located on each main landing gear beam. These signals are used to configure the airplane systems to the appropriate air or ground status.

The airplane is equipped with nose wheel steering and main gear aft axle steering. Nose wheel steering is powered by the centre/reserve hydraulic system. Primary steering control is provided by a nose wheel steering tiller for each pilot. Limited steering control is available through the rudder pedals. The tillers can turn the nose wheels up to 70 degrees in either direction. A pointer on the tiller assembly shows tiller position relative to the neutral setting. The rudder pedals can be used to turn the nose wheels op to 7 degrees in either direction. Tiller inputs override rudder pedal inputs. Main gear aft axle steering automatically operates when the nose wheel steering angle exceeds 13 degrees to reduce tire scrubbing.

Each main gear wheel has a multiple disc carbon brake. The nose wheels have no brakes. The brake system includes:

- Normal brake hydraulic system
- Alternate/reserve brake hydraulic system
- Brake accumulator
- Antiskid protection
- Autobrake system
- Parking brake

The normal brake hydraulic system is powered by the right hydraulic system. The brake pedals provide independent control of the left and right brakes.

Alternate/reserve brake hydraulic system selection is automatic. If the right hydraulic system pressure is low, the centre/reserve hydraulic system automatically supplies pressure to the alternate/reserve brake hydraulic system.

The brake accumulator is located in the normal brake hydraulic system. If right and centre/reserve brake hydraulic power is lost, the brake accumulator can provide several braking applications or parking brake application.

Antiskid protection is provided in the normal and alternate/reserve brake hydraulic systems. Antiskid protection is also provided when the brake system is being supplied pressure only from the brake accumulator. The normal brake hydraulic system provides each main gear wheel with individual antiskid protection. When a wheel speed sensor detects a skid, the associated antiskid valve reduces brake pressure until the skidding stops. The alternate/reserve brake hydraulic system provides antiskid protection to tandem wheel pairs for the forward and middle axle wheels. The aft axle wheels remain individually controlled.

The autobrake system provides automatic braking at preselected deceleration rates for landing and full pressure for rejected takeoff. The system operates only when the normal brake system is functioning. Antiskid system protection is provided during autobrake operation.

The parking brake can be set with the normal or alternate brake hydraulic system pressurized. If the normal and alternate brake systems are not pressurized, parking brake pressure is maintained by the brake accumulator. The parking brake is set by depressing both brake pedals fully, while simultaneously pulling the parking brake lever up. This mechanically latches the pedals in the depressed position and commands the parking brake valve to close. The parking brake is released by depressing the pedals until the parking brake lever releases.

Warning Systems

Warning systems consists of:

- Engine Indication and Crew Alerting System (EICAS)
- Airspeed alerts

- Tail strike detection system
- Takeoff and landing configuration warning system
- MCP selected altitude alerts
- Crew alertness monitor
- Terminal Collision Alerting System (TCAS)
- Windshear alerts
- Ground Proximity Warning System (GPWS)

EICAS consolidates engine an airplane system indications and is the primary means of displaying system indications and alerts to the flight crew. The most important indications are displayed on EICAS, which is normally displayed on the upper centre display. System conditions and configuration information is provided to the crew by four types of EICAS messages:

- EICAS alert messages are the primary method to alert the crew to non-normal conditions
- EICAS communication messages alert the crew to communication messages received or sent
- EICAS memo messages are crew reminders of certain flight crew selected normal conditions
- EICAS status messages indicate equipment faults, which may affect airplane dispatch capability

There are a four airspeed alerts. They are:

- Stall Warning
- Airspeed Low
- Takeoff V1 Airspeed
- Overspeed Warning

Stall Warning is a warning of an impending stall, which is provided by left and right stick shakers, which independently vibrate the left and right control columns. AIRSPEED LOW is displayed when airspeed is below manoeuvring speed. The voice annunciation VEE ONE sounds when airspeed reaches V1 during takeoff. OVERSPEED is displayed when airspeed is greater than manoeuvring speed.

The tail strike alert system detects ground contact, which could damage the airplane pressure hull. The system consists of a two-inch blade target and two proximity sensors, and is installed on the aft body of the airplane.

The takeoff and landing configuration warning system alerts the crew that the airplane is not configured for normal takeoff or normal landing.

Altitude alerting is provided when approaching or departing the altitude selected in the MCP altitude window.

TCAS (Terminal Collision Alerting System) alerts the crew to possible conflicting traffic. TCAS interrogates operating transponders in other airplanes, tracks the other airplane by analyzing the transponder replies, and predicts the flight paths and positions. TCAS provides TCAS ND messages, voice annunciation, PFD vertical flight path guidance, and traffic displays of the other airplanes to the flight crew. TCAS operation is independent of ground-based air traffic control. TCAS identifies a three-dimensional airspace around the airplane where a high likelihood of traffic conflict exists. The dimensions of this airspace are contingent upon the closure rate with conflicting traffic.

GPWS provides immediate alerts, and look-ahead obstacle and terrain alerts for potentially hazardous flight conditions involving imminent impact with the obstacles and the ground. GPWS alerts are based on radio altitude, barometric altitude, ADIRS, glide slope deviation, and airplane configuration. GPWS alerts are provided for the following:

- Altitude loss after takeoff or go-around
- Excessive and severe descent rate
- Excessive terrain closure rate
- Unsafe terrain closing clearance when not in the landing configuration
- Excessive deviation below ILS glide slope
- Altitude advisories
- Immediate windshear

GPWS also provides look-ahead terrain mode alerts by monitoring obstacle and terrain proximity using a worldwide terrain database and an obstacle database. If there is a potential obstacle or terrain hazard, GPWS look-ahead alerts are provided based on estimated time to impact. Estimated time to impact is based on airplane position, barometric altitude, present track, vertical path, and ground speed.

2.2.2 Expert Systems [3]

2.2.2.1 What Is an Expert System?

An expert system is a computer program designed to model the problem-solving ability of a human expert.

The two major traits of an expert that are modelled in an expert system are the expert's knowledge and reasoning. In order to accomplish this, the system must have two principles modules: a knowledge base and an inference engine.

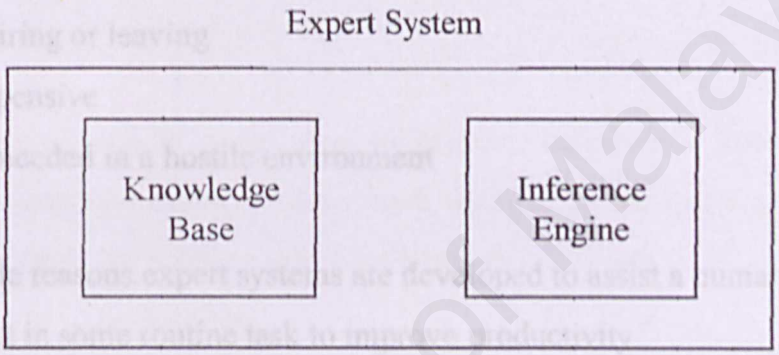


Figure 2.3 Expert System Block Diagram

The knowledge base contains highly specialized knowledge on the problem area as provided by the expert. It includes problem facts, rules, concepts, and relationships.

The inference engine is the knowledge processor, which is modelled after the expert's reasoning. The engine works with available information on a given problem, coupled with the knowledge stored in the knowledge base, to draw conclusions or recommendations.

2.2.2.2 Why Build an Expert System?

Experts are a valuable resource to an organization and are vital in enhancing its productivity because they can offer creative ideas in alleviating problems, while accomplishing routine tasks. Expert systems, which have the problem-solving capabilities of a human expert, seem to have

added advantages compared to their human counterparts. It can produce continuous support with consistent performance in a fraction of the time it takes a human expert to complete the same task. It can be easily and economically duplicated for distribution to locations lacking the expertise, and can even be operated in harmful environments. The main motivations behind the need for expert systems are:

- Replacement of an expert
- Assisting an expert

Some of the principle reasons expert systems are developed to replace a human expert are:

- Make available expertise after hours or in other locations
- Automate a routine task requiring an expert
- Expert is retiring or leaving
- Expert is expensive
- Expertise is needed in a hostile environment

Some of the principle reasons expert systems are developed to assist a human expert are:

- Aiding expert in some routine task to improve productivity
- Aiding expert in some difficult task to effectively manage complexities
- Making available to the expert, information that is difficult to recall

2.2.2.3 Where Have Expert Systems Been Built?

There are very clear signs that experts systems are merging with the mainstream of information processing. The field has developed tremendously and this is apparent in the acceptance of the technology by the commercial sectors. Some of the major application areas of expert systems are:

- Medicine
- Business
- Manufacturing
- Engineering
- Power Systems

2.2.2.4 How Are Expert Systems Used?

Expert system applications can be categorized by its problem-solving paradigms. They are:

Control

Control systems adaptively govern the behaviour of a given system. An expert control system obtains data on the system's operation, interprets the data to form an understanding of the state of the system or a prediction of its future state, and determines and executes needed adjustments. Control systems must also perform monitoring and interpretation tasks to track system behaviour over time.

Design

Design systems configure objects under a set of problem constraints. These systems usually perform their tasks following a series of steps, each with their own specific constraints. These steps are usually dependent upon each other, which makes it difficult to assess the impact that a change in one step will have on another.

Diagnosis

Diagnosis systems infer systems malfunctions or faults from observable information. Most diagnosis systems have knowledge of possible fault conditions with means to infer whether the fault exists from information on the system's observable behaviour. A more recent trend in the field relies on a model-based reasoning approach, which models the system's normal behaviour, and detects and diagnoses faults from deviations in expectations. Most diagnosis systems include a prescription task that offers a remedy to the detected fault.

Instruction

Instruction systems guide the education of a student in a given topic. They treat the student as a system that must be diagnosed and repaired. Typically, they begin by interacting with the student

to form a model of the student's understanding of the topic. They then compare the student model with an ideal model to uncover weaknesses in the student's understanding. This task is then followed by remedial instruction to correct any misunderstanding.

Interpretation

Interpretation systems produce an understanding of a system from available information. Typically, this information consists of data from such sources as sensors, instruments, test results, etc. these systems translate the raw data into symbolic form that describes the situation.

Monitoring

Monitoring systems compare observable information on the behaviour of a system with systems that are considered crucial to its operation. Monitoring systems will usually interpret signals from sensors and compare the information with known crucial states. When a crucial state is detected, an established sequence of tasks will be performed.

Planning

Planning systems form actions to achieve a given goal under problem constraints. Some planning systems must have the flexibility to change the series of planned tasks when they obtain new problem information. To accomplish this, they need the ability to backtrack and reject a current line of reasoning in favour of exploring a better one.

Prediction

Prediction systems infer likely consequences from a given situation. These systems attempt to predict future events using available information and a model of the problem. Prediction systems often must be able to reason about time or ordered events. Models must be available to infer how some given actions can influence future events. Intelligent simulation codes are often used in these types of systems.

Prescription

Prescription systems recommend solutions to a given system malfunction. These types of systems usually first incorporate a diagnostic task to determine the nature of the malfunction. More advanced systems incorporate planning and prediction techniques to create tailored remedies.

Selection

Selection systems identify the best choice from a list of possibilities. They work from problem specifications defined by the user and attempt to find a solution that matches these specifications most closely. These systems usually employ an inexact reasoning technique or a matching evaluation function when forming their selections.

Simulation

Simulation systems model a process or a system to permit operational studies under various conditions. They model the various components of the system and their interactions. Using the model along with the user-supplied information, these systems can be used to predict operating conditions for the real system.

2.2.2.4 Characteristics of an Expert System

Separates Knowledge from Control

Separating the system's knowledge (knowledge base) from its control (inference engine) is a characteristic of an expert system that distinguishes it form a conventional program. This makes the task of modifying and maintaining the system easier. Changes or additions of pieces of knowledge to the knowledge base and modifications to the inference engine's algorithm can be done easily.

Possesses Expert Knowledge

An expert system contains knowledge that exemplifies the expertise of a human expert. The expertise that is captured and encoded in an expert system includes both domain knowledge and problem solving skills.

Focuses Expertise

An expert system has a narrow area of expertise, which makes it proficient at what it knows but performs poorly outside its area of expertise. Successful expert systems are those that have a well-focused problem domain.

Reasons with Symbols

An expert system embodies knowledge such as facts, concepts and rules in symbolic form. Besides representing statements in symbolic form, an expert system can also manipulate these symbols when solving a problem to make logical conclusions.

Reasons Heuristically

Human experts can solve problems efficiently using their experiences. Their experiences help them form a better understanding of the problem, which they retain in form of rules-of-thumb or heuristics. This reasoning strategy can be replicated in an expert system by not using strict procedures found in conventional programs.

Permits Inexact Reasoning

An expert system allows the capturing and encoding of information that is uncertain, ambiguous or unavailable, and by domain knowledge that is inherently inexact.

Is Limited to Solvable Problems

Expert systems can only apply to problems that human experts can solve. Therefore, expert systems should not be designed for application in areas where the problem is new or frequently changing.

Thrives on Reasonable Complexity

The problem to be solved should not be too simple or too complex. A problem that takes too long to solve is probably beyond the capability of the expert system. One way to approach these kinds of problems is to break them into sub-topics, each of which can be solved with a single expert system.

Makes Mistakes

Expert systems, like their human counterparts, can make mistakes. This cannot be seen as a disadvantage compared to conventional programming. This is because, if there is faulty or missing data, conventional programs may produce no results at all but expert systems can still arrive at a reasonable conclusion if not the exact one.

2.2.3 Sample System [6]

2.2.3.1 Intelligent Aircraft/Airspace System (IAAS)

Abstract

The worldwide Aircraft/ Airspace System (AAS) is faced with a large increase in air traffic in the coming decades, yet many flights already experience delays. The AAS is comprised of many different agents, such as aircraft, airlines, and traffic control units. Technology development will make all the agents in the AAS more intelligent; hence, there will be an increasing overlap of the declarative functions of the agents. The Intelligent Aircraft/Airspace System (IAAS) provides improved system performance, redundancy, and safety by utilizing the overlapping capabilities of the agent. Principled negotiation between agents allows all the agents in the system to benefit from multi independent declarative analysis of the same analysis. Multi-attribute utility theory and decision trees are used as the basis for analyzing the behaviour of different types of agents. Intelligent agents are modelled as rule-based expert systems whose side effects are the procedural and reflexive functions of the agent. Principled negotiation is also the side effect of the expert system's declarative functions. A hierarchical organization of agents in the IAAS is proposed to facilitate negotiation and to maintain clear lines of authority.

Behaviour of Intelligent Agents

To model the behaviour of a collection of Intelligent Agents, we need to understand the decision-making and control processes of each agent. We can use declarative, procedural, and reflexive, functions to describe the behaviour of any Intelligent Agent operating within the AAS. Traffic Management Agents (TrMAs) and Aircraft are two types of AAS agent. There is great similarity between the declarative functions of these two agents and of all agents in the IAAS. All agents identify scenarios, assess the situation, and then make decisions based on that assessment. All agents operate in the same environment, but they have different interests and priorities when choosing how to respond to particular situations.

Table 2.1 Function Hierarchies for Two Agents

Function	Traffic Management Agent	Aircraft Agent
Declarative	Traffic monitoring and prediction	System monitoring
	Conflict detection and prediction	Goal planning
	Constraint monitoring	Scenario identification
	Hazard detection	Choice of operating mode
	Weather monitoring and prediction	Conflict resolution
	Assessment of pilot requests	
	Trajectory assignment	
Procedural	Conflict resolution	Adaptation
	Trajectory adaptation	Guidance and navigation
	Flow control	Estimation and control
	Communications	Crew coordination
		Communications
Reflexive	Display update	Measurement
	State vector processing	Actuation
	Aircraft handover	Inner-loop regulation
	Flight information processing	

Model of a Negotiating Intelligent Agent

The declarative functions of intelligent agents are well modelled by expert systems. Goal planning, scenario identification, and operating mode selection require reasoning. Alternatives must be evaluated, and decisions must be made through a process of deduction, that is, by inferring answers from general or domain-specific principles. A rule-based expert system uses an inference engine to process the knowledge and beliefs contained in its rule base and database. The procedural and reflexive functions of the agent, such as communication and estimation, are side effects of a rule being queried by the expert system. As well as providing a model for analysis and collective intelligent agent behaviour, structure could be used as a basis for systems that provide intelligent assistance to humans (pilots, controllers, and schedulers) or for systems that operate autonomously under human supervision. The negotiation functions are carried out as

side effects of the expert system. The expert system deduces the weighting function, constraints, and heuristics that are used by the systems that generate and assess options.

The functions of an Intelligent Agent can be divided into four task groups: emergency tasks, mode specific tasks, negotiation tasks, and routine tasks. The top-level structure of the rule base controlling these task groups is shown Figure 2.4. Rectangular boxes represent parameters, and ovals represent rules. Parameters contain raw data and other factual information about the domain. The possible values of the parameter are shown in the slots. Rules describe the relationship between the parameters. The form of the rule is shown by the line linking the parameters and the rules. AND relationships are indicated by arcs between the lines.

The inference engine traverses the rule base using backward and forward-chaining. If the value of an unknown parameter is desired, a goal-directed search (backward-chaining) is used. The logical outcome of the new data, or a set of premises, is found by a data-driven search (forward-chaining). The control cycle is driven by backward –chaining to establish a value of a top-level parameter. This fires Rule 1, which is read:

IF the value of parameter EMERGENCY TASK COMPLETED has been determined
AND the value of parameter MODE SPECIFIC TASKS COMPLETED is TRUE
AND the value of parameter NEGOTIATION TASKS COMPLETED has been determined
AND the value of parameter ROUTINE TASKS COMPLETED is TRUE
THEN set the value of parameter TOP-LEVEL SEARCH COMPLETED to TRUE

EMERGENCY TASK COMPLETED will be set to TRUE only if the actions that are required in an emergency situation have been executed. The value of the parameter may be TRUE or FALSE; the agent is not in an emergency situation at all times. The use of the shaded box indicates that the search could continue once the value of EMERGENCY TASK COMPLETED has been determined, whether it is TRUE or FALSE.

Mode specific tasks are functions that are executed only f the agent is in a particular state. For example, an action such as TAXI INSTRUCTIONS LOADED would be required only when an aircraft was on the ground or on approach; it would not be executed on departure or en-route phases.

The routine tasks are undertaken whatever the mode of an agent. A TrMA will update radar displays, fuse raw data, and update databases on each control cycle.

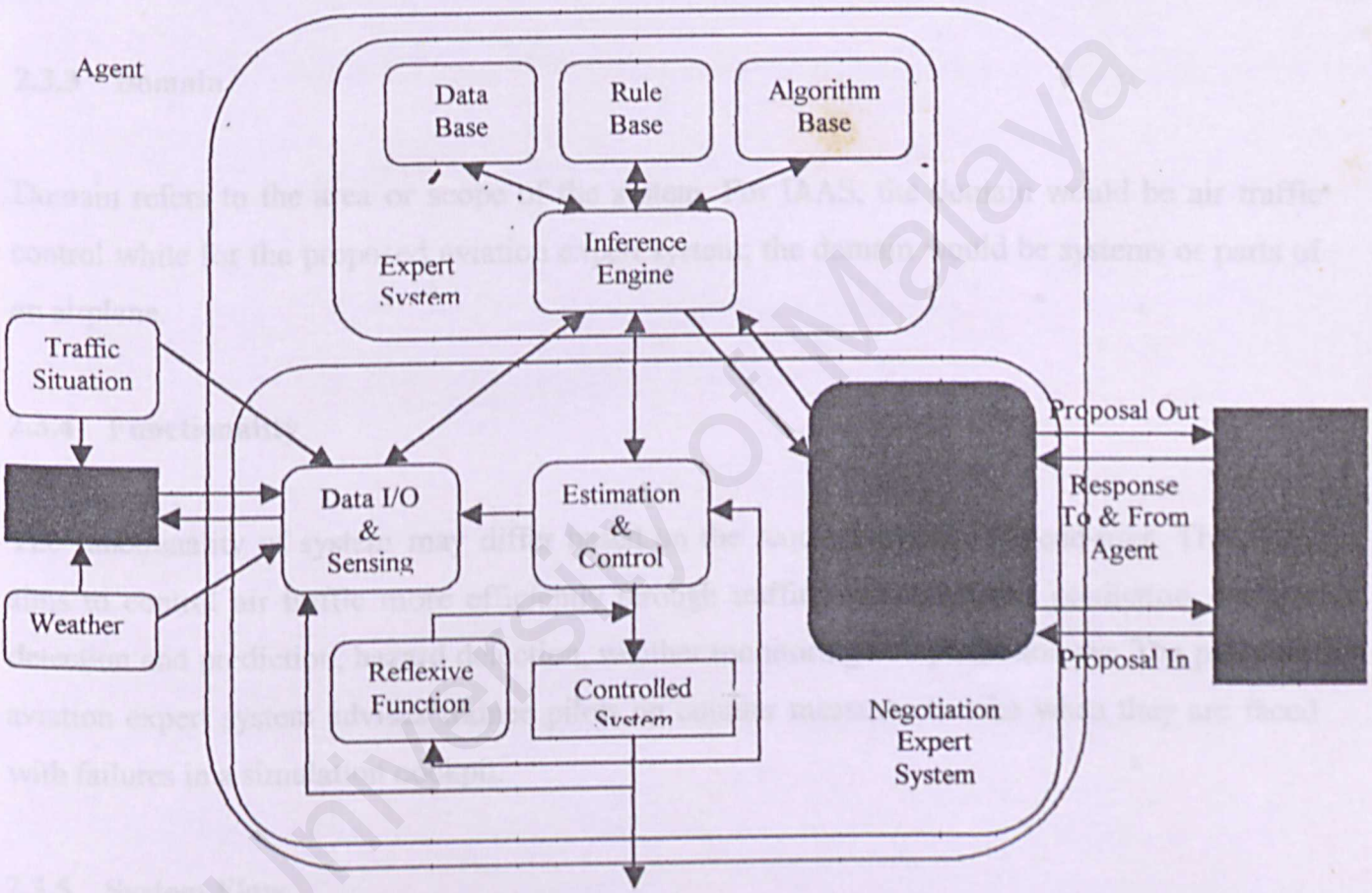


Figure 2.4 Structure of an Intelligent Agent

2.3 Analysis

The system that was reviewed can be used as a guideline to develop a rule-based expert system. The analysis conducted on the system can be broken up into four main parts, they are:

- Domain
- Functionality
- System Flow
- Implementation

2.3.3 Domain

Domain refers to the area or scope of the system. For IAAS, the domain would be air traffic control while for the proposed aviation expert system; the domain would be systems or parts of an airplane.

2.3.4 Functionality

The functionality of system may differ based on the requirements of the end-user. The IAAS aims to control air traffic more efficiently through traffic monitoring and prediction, conflict detection and prediction, hazard detection, weather monitoring and prediction etc. The proposed aviation expert system advises trainee pilots on counter measures to take when they are faced with failures in a simulation cockpit.

2.3.5 System Flow

System flow is the way an application works. Basically, the IAAS expert system and the proposed aviation expert system have the same system flow. The end user will input information into the system based on distress signals, alert messages or observable environs and this information will be compared to that of in the knowledge base. Rule-based reasoning techniques are used to produce results for the end-user.

2.3.6 Implementation

Like the IAAS, the proposed aviation expert system will utilise a rule-based system for the approach on problem solving. An intelligent system that incorporates artificial intelligence (AI) would require development using AI languages. The most popular AI languages in the market at the moment are Win-Prolog and Visual Prolog, which implement programming in logic. The author plans to use Visual Prolog 5.2 to develop the proposed aviation expert system.

2.4 Development Tools

The proposed system is an intelligent system that incorporates artificial intelligence. For an AI system, it is strongly recommended by researches and most software engineers that the Prolog language be used. Prolog is short for programming in logic. It has many features required by intelligent systems that conventional development languages do not have. Below is a list of these features that led to the usage of Prolog as the development tool [7].

- Prolog is descriptive
- Prolog uses facts and rules
- Prolog can make deductions
- Prolog program execution is controlled automatically
- Prolog has a very short and simple syntax
- Prolog is powerful

Prolog is descriptive

Most conventional languages would specify a series of steps on how the computer must work to solve a problem. Prolog is different. It contains a description of the problem. The description consists of two components. The first one is the description of the objects involved in the problem [7]. The second component consists of facts and rules describing the relations between the objects.

Prolog uses facts and rules

Prolog consists of facts and rules. Facts are like:

mother (mary, john)

This statement means that Mary is the mother of John. Rules on the other hand are like:

brother (A, B): - mother(C, A), mother(C, B)

The following rule states that A is the brother of B if C is the mother of A and C is the mother of B.

To define the same set of rules in other conventional languages would involve several lines of coding.

Prolog can make deductions

What this means is that when you give Prolog a set of rules and a goal, it is able to come up with an answer to the goal. For example:

Goal mother (mary, john).

Prolog will answer, "yes" to this goal. This is because the goal matches a stored fact. If a variable is used in place of John's mother's name, Prolog will find the value for the variable. For example:

Goal mother (X, john).

Prolog will answer X = Mary because Mary satisfies the rule for mother. The solutions to these goals are derived through backtracking. The automatic backtracking mechanism is Prolog's most powerful feature.

Prolog program execution is controlled automatically

When a goal is executed, Prolog tries to find all possible values that satisfy the given goal [7]. With the backtracking mechanism, once a solution has been found, it retraces through the stored facts to see if other variables may provide a solution to the goal.

Prolog has a very short and simple syntax

When compared to other conventional languages, Prolog has a much shorter syntax. This makes Prolog easier to learn and use compared to other languages.

Prolog is powerful

Prolog is said to be a higher-level language compared to other languages, such as C or Pascal. Prolog uses one tenth as many program lines when solving a problem compared to C or Pascal.

The fact that Prolog has built-in pattern recognition facility makes Prolog a far more powerful tool than other languages. Besides pattern recognition, the ability to handle recursive structures in a simple and efficient way makes Prolog stand out.

Thus, the author considers Prolog the most practical language to be employed as a development tool. Now that Prolog has been chosen as the development tool, the choice of choosing between Visual Prolog and WIN-PROLOG arises. Therefore, the author studies both languages to determine the best and most suitable language for the development of his system.

2.4.3 Visual Prolog 5.1

Visual Prolog is the outcome of Prolog Development Center (PDC) after years of research work. Visual Prolog has increasingly become the tool of choice for many system developers, because of the intelligent features that can be readily added to programs or even web sites. Visual Prolog addresses the same target market as SQL Database Systems, C++ development systems and other languages like Visual Basic. An application developed in Visual Prolog can have far superior performance and user friendliness along with shorter development time.

PDC's Prolog has been found to be very well suited when it comes to traditional database tasks. This is because Visual Prolog has a very easy-to-use database engine. Visual Prolog programs are fast due to its compiler, which happens to perform as well as C++ compilers. Web support and object system contribute towards the usefulness of Prolog in the commercial sector.

Visual Development Environment (VDE)

The VDE combines the compiler with various components [7]. These components are an editor, a resource and application Expert, interactive producing facilities and various browsing facilities. A running prototype is automatically generated once the user interface components are created. The application Expert creates all the necessary files for a project and the resource Expert generates the Prolog code to support all the selected resources.

Visual Prolog's VDE is designed to make it easier, faster and more convenient to develop applications based on a higher level abstraction of the standard interfaces provided by each of the native operating systems.

Code Expert

A Code Expert is a feature in Visual Prolog that creates and maintains the Prolog control code for the resources. The combination of the Layout tools and Code Expert is said to be one of Prolog's greatest strengths. The function of the Code Expert is to reduce your workload by

allowing you to create a new application in only a few minutes and incrementally enhance this form of prototype to your final application.

An Application Expert that generates and configures projects

An application expert can generate a new project. It accounts for thousands of combinations of Operating systems, UI strategies, C compilers, companion tools and so on. When a new project is generated, it will automatically set up all the basic tools like a help file, toolbars and menus [7].

Integrated editors for the preparation of resources

These tools make it possible to visually design and modify the user interface in an interactive way. The mouse may be used to lay out controls in dialogs or windows and access attribute settings with a mere click. Resources consist of Windows, Dialogs, Bitmaps, Icons, Cursors and Strings that are necessary for any application that uses a GUI (Graphic User Interface).

Ability to import resources

Visual Prolog has the ability to import resources from DLLs, EXEs, RES files and resources from other Visual Prolog projects. This makes Visual Prolog very flexible to use and to be integrated with other languages.

A language-sensitive text editor

Most languages have text editors for codes to be written in. Like other languages, Visual Prolog has a powerful source code editor with colour coding of Visual Prolog keywords and other language elements. The colour makes it much easier to differentiate between names, parameters, comments, etc. The editor supports unlimited undo and redo facilities, search and replace, cut, copy, paste, drag and drop for quick block movements, and permits embedding of hypertext links.

Integrated state-of-the-art Help Maker

There is a built-in help authoring system that makes it very easy to give your application online help. The Help system is based on the PDC Hypertext Abstract Machine (HAM). In this system it is possible to enter the help text interactively, mark new links with the mouse and to follow existing links during the design phase. The help system is able to output both the Windows .RTF format and the OS/2 IPF format so it is possible to generate native help systems for both OS/2 and windows.

Visual Programming Interface (VPI)

Visual Prolog has a predefined portable Prolog API for graphical user interface. This API is an abstraction of the facilities found in the basic windowing environments of the Windows 3.x, Windows 95, Windows NT, OS/2 PM platforms. This gives the Visual Prolog programmer an API that is both portable and easier to use than programming to the native API. However, in order to restrict users, the API also contains platform-specific facilities and options that are not portable.

High-level GUI components

A number of high-level GUI components have been implemented on top of the VPI. These components are supplied with source code and are of course portable to all the platforms supported by the VPI. These tools include a Grid, a Tree Window, an Explorer view, Toolbars, Tabbed dialogs, advanced report handling and so on.

Debugger

A debugger is one tool in Visual Prolog that cannot be set aside. The debugger works on the compiled code and allows the setting of breakpoints and stepping through the code. While this is done, variable values and the contents of the asserted facts can be inspected.

Exception handling and error handling

Visual Prolog includes powerful mechanisms for handling error situations, as well as controlling user breaks. The programmer may select various levels of error checking and error reporting. Below is an example of the error checking function that checks a diskette for error:

```
Check_diskette (A): -
        Trap (disk (A), ExitCode, errorhandler (ExitCode)).
```

A powerful database subsystem

Visual Prolog has a very fast and flexible database subsystem. This feature makes Visual Prolog a more suitable choice for database applications compared to many other 4GLs. The database subsystem supports a collection of distinct ordered chains of Prolog terms, where database terms can be any abstraction supported by the language itself, from simple records to trees or graphs. The database system can directly access individual terms, or it can backtrack through chains of terms to generate or match particular values. These terms can be stored in any one of these three locations. The first one is in a file, the second in memory and the third in EMS-type expanded memory under DOS. The database also supports B+ trees, which provide fast data retrieval and the ability to change term ordering efficiently.

Client-Server architecture

Visual Prolog is a powerful platform for building Client-Sever applications. The main avenue for this is currently the TCP/IP bindings, but under the OS/2 it is also possible to use Named Pipes and NETDDE, which are available in Windows [7]. Using any of these facilities, the programmer can send arbitrarily complex Prolog terms between multiple processes on a single machine, or between programs on separate machines over a network. Database or logic servers can easily be constructed with this facility.

ODBC and portable SQL Bindings

Visual Prolog has portable SQL bindings that allow it to link programs to external databases when needed. The portable SQL bindings are based on either ODBC, Oracle's OCI libraries or DB2 under OS/2. Visual Prolog also maintains more extensive direct bindings to Microsoft's ODBC API's for Windows platforms.

Document handling Tools

PDC's DOC tools provide a high-level abstraction for handling richly formatted documents. Using a Prolog structure to represent the document makes it possible to be independent of the actual format whether it is >RTF, HTML or >IPF. There are generators to convert the Prolog term format to these formats, as well as parsers to convert any of these formats to the Prolog term format. These tools open up many application possibilities, like the generation of Word documents, Internet assistants and so on.

FTP and HTTP support

Visual Prolog has the ability to use the Internet File Transfer Protocol to send and receive files from an Internet server. It has APIs that support Hypertext Transfer Protocol, which is the basic protocol used by the World Wide Web. These APIs can be used to create WWW client and server utilities as well as Internet Agents in Visual Prolog.

2.4.4 WIN-PROLOG

WIN-PROLOG is a product of Logic Programming Associates Ltd (LPA). LPA's previous version of Prolog was the LPA-PROLOG, which provided a 32bit-programming environment within the Win16 API of Windows 3.1. Now, LPA has introduced a new version called WIN-PROLOG, which is a true Win32 application, working directly with the 32-bit Windows NT and Windows 95 API [8].

WIN-PROLOG is the central product in a series that consists of programming tools that works cross-platform on Windows 2000, NT, ME, 98, 95 and 3.1; the series also includes flex, Prolog++, the Portable Dialog Manager, and the ProData Database Interface toolkit. These products have access to all the memory available to Windows without restriction. The Windows series uses incremental compilation of user programs to provide the execution speed of a compiler but with the interactive behaviour of an interpreter. The optimizing compiler provides multiple-argument indexing for static code.

WIN-PROLOG shares the same underlying 32-bit inference engine and architecture as LPA DOS-PROLOG and LPA MacProlog32, and is closely compatible with both the ISO Prolog and Quintus Prolog for Unix. There is a powerful bi-directional DLL interface which allows code written in C, C++, Pascal etc to be accessed. It also has a configurable DDE for linking with other applications (Excel, Word, Visual Basic etc). WIN-PROLOG includes full Unicode support, an attractive multi-window development environment, interactive source-level debugger, integrated editor and high-level access to Windows GUI functions.

Flex is a powerful hybrid expert system toolkit with frames, instances, daemons, rules, relations, questions, multiple inheritance, dynamic rule sets and has full access to Prolog. Flex contains its own Knowledge Specification Language (KSL), whereby rules and frames are defined in an English-like natural language.

The Prolog++ toolkit is a complete OOP language with dynamic and static objects, instances, methods and attributes which supports multiple inheritance, message cascading and has full access to Prolog. To aid program development there is a graphical object hierarchy browser.

WIN-PROLOG: for Windows 2000, NT, ME, 98, 95 and 3.1

WIN-PROLOG is a true 32-bit application, which runs under the older Windows 3.1 systems as well as having specially designed enhancements for Windows 2000. It is totally integrated with the Windows operating system, and utilizes a Multiple Document Interface (MDI) environment for editing, compiling, debugging and running programs.

The development environment permits any number of program edit windows to be open at any one time, and you can cut and paste text between them. Standard search and replace edit facilities; together with special Prolog-related ones (such as "goto definition") make the maintenance of large and complex systems easy [8].

You can compile or optimize program edit windows individually or all together, and save your work when you want to. A full source-level debugger utilizes a special dialog in which you can scroll through program source code, variable bindings, and other information. A multi-level break facility allows you to escape from the debugger to run supplementary queries before returning. A traditional box model debugger, together with a collection of small, special purpose debuggers, complements the source-level debugger to provide unprecedented flexibility in program testing.

Unicode and Other Text Formats

WIN-PROLOG manages to support a full 32-bit character set, and hence Unicode, ISO/IEC 8859-1 and ASCII, with minimal space or processing overheads compared to previous 8-bit character support by employing a custom text encoding format, known as "UTF-BS". Character-level filtering of input and output is supported on a file-by-file basis, allowing applications to handle multiple character sets simultaneously and transparently, in any of 11 file

formats, including ASCII, ISO/IEC 8859-1, UTF-8, UTF-16BE, UTF-16LE, UTF-32BE and UTF-32LE among others. External API calls can be made with text in four formats, including WIDE (Windows Unicode), ANSI (Windows Multibyte) and ISO/IEC 8859-1. Altogether, WIN-PROLOG's text handling is second to none.

The Windows Interface

WIN-PROLOG has the ability to directly program and control most of the features available in the Windows GUI. Using built in predicates; one can create and destroy windows, model and modeless dialogs, control items, menus and fonts. Windows can be renamed, resized, repositioned, hidden or shown, enabled or disabled. List boxes, combo boxes and menus can be added to, deleted from and inserted into. Text in edit controls can be selected by offset or line number, searched, scrolled, replaced, deleted and inserted into. Buttons can be restyled, grouped, checked or unchecked, selected or deselected, grayed or enabled. Fonts of any face, size and style can be applied to any control window, including edit, button, static, list box and combo box controls. Initialization files can be created, written, read, added to and deleted from.

Links can be found from any given window to its parents, siblings and children, and top-level windows can be searched for by class and name. All of the above operations can be applied to any window whose handle is known. You can read or write text from and to any window, thanks to the special string data type, the passing of large amounts of data between Prolog and Windows is both simple and efficient.

Powerful Graphics Functions

WIN-PROLOG includes a comprehensive and carefully integrated Graphics subsystem. This provides both the graphics functions needed to draw lines, ellipses, rectangles, polygons, bitmaps, icons, metafiles and cursors, and those needed to maintain resources. Each graphic resource, is given a Prolog name which is maintained, automatically, in a dictionary. When such a resource is no longer required, it can be deleted simply to free up system memory. Icons, bitmaps and metafiles can be loaded direct from disk, and in the case of icons, extracted from

any Windows executable file. Metafiles can also be created within WIN-PROLOG for exporting to other applications, such as Word for Windows or CorelDraw!

The repainting of graphics windows is handled elegantly through the use of message handlers and all necessary housekeeping is carried out "behind the scenes". Graphics can be painted into any window, but special support is given both for buttons (allowing the creation of graphical toolboxes) and special "grafix" control windows [8].

Message Handlers and Hooks

WIN-PROLOG has the ability to create and manipulate highly complex dialogues and menus. It also includes sophisticated ways to handle the messages generated by these and other GUI objects. Once you have created a dialogue, filled it with controls, and placed initial data into them, you can use the dialogue as often as you wish, either in a model or modeless fashion. All button clicks, list and combo box selections, edit control changes, and scroll bar movements are reported to a special handler program whose job it is to process the dialogue. Each dialogue can have its own handler, or they can share handlers. The handlers are written in Prolog to perform the function that you need.

The same way, when you have built a menu, with or without submenus, you can install it on the main menu bar. Whenever selections are made, messages are sent to a special menu handler. As well as the user dialogue and menu handlers, several message "hooks" are provided to enable you to capture and process certain messages from the predefined modeless dialogues, DLL modules and the keyboard.

Direct Windows API Interface

WIN-PROLOG has a special interface which allows your programs to call virtually any C function directly, whether defined in the Windows API or in a DLL or other module. The winapi/3 predicate allows any function, defined in any 32-bit module, to be called with any number of parameters. The parameters may be integers, string pointers, or arbitrary structures,

and facilities exist for defining named "memory files". All memory allocation and stack frame creation is carried out automatically.

The Windows API function means that your programs are no longer "limited" to using the 100-plus API functions built into WIN-PROLOG by LPA, and you can add additional functions yourself as and when you need them. It also provides a direct way to invoke code in DLLs, without having to write parameter-translating ("glue") code in C/C++.

Memory Files

WIN-PROLOG has the powerful ability to open "virtual" files in memory. These have no disk image, and can be of any size within available resources. Ideal as temporary scratchpads during complex data operations, they are extremely fast and avoid the usual worries about how to name a temporary file best.

Modeled on the original buffers that were implemented to provide storage for use in conjunction with the winapi/3 predicate. These new memory files can be used with any and all of the standard Prolog input/output predicates, as well as to store virtually any data directly. Applications that require the storage of large numbers of small data items (such as bit tables) are especially assisted by this feature.

True 32-bit Assembler-coded Implementation

The WIN-PROLOG kernel is implemented entirely in 32-bit assembler to provide the best overall performance possible on 386, 486, Pentium or Pentium Pro platforms, with just enough 32-bit C to interface it to the Win32 API of Windows 2000, NT, 98 and 95. WIN-PROLOG will also run on Windows 3.1 systems, provided that these are equipped with the Microsoft Win32s subsystem. Thanks to its tight implementation, the entire WIN-PROLOG system requires under 2Mb of disk space, and can be installed without having to modify any Windows or System directories.

Thorough Quintus Prolog Compatibility

WIN-PROLOG has been designed from scratch for Quintus Prolog (QP) compatibility. This extends well beyond the obvious requirement of duplicating the built in predicates of QP, as it includes special features such as logical file names, background housekeeping of predicates/file relationships, and much more. Most applications will port directly from QP to WIN-PROLOG, as the file management support isolates user programs from the intricacies of the host operating system.

Powerful Input and Output Features

WIN-PROLOG's has a great ability in its powerful collection of input and output subsystems. As well as the standard see/1, tell/1, read/1 and write/1 style predicates, support is given for formatted I/O, binary I/O, and the manipulation of a number of special I/O streams, including device and string streams as well as disk files.

Formatted I/O primitives provide complete control of the output of atoms, strings, numbers and other items. Fixed field display, with left or right justification, optional truncation, and free field formats are all supported. Numbers can be output in fixed point, signed or unsigned integer, and even non-decimal base formats (anything from base 2 (binary), through base 16 (hexadecimal) right up to base 36).

Matching formatted input allows structured records to be read in with automatic type checking. As well as the obvious applications such as writing or reading data in tabular formats, the formatted I/O predicates make it easy to build interfaces to ASCII files from other applications.

The Operating System Interface

WIN-PROLOG has a huge library of operating system interface functions, providing for disk file and directory management, program execution, the reading of environment variable strings, and time and date functions. Predicates allow you to create, rename and remove files or directories,

and you can test and modify file attributes and timestamps. File directories can be read according to file name and attribute matching, or returning information about each file's size, time and date.

For specialist time and date applications, a built in predicate allows you to compute absolute day numbers for any given date, or vice versa, allowing days between dates, lunar phases, and other such calculations.

2.4.5 Analysis

In this section, the author decides on whether to use Visual Prolog or WIN-PROLOG as the development tool. The author has concluded, after an analysis on both languages, that Visual Prolog has better prospects than WIN-PROLOG. These are the reasons:

- Visual Prolog has an Application Expert that creates all the necessary files for a project. It also automatically sets up all the basic tools like a help file, toolbars and menus when a new project is generated.
- Visual Prolog has a Code Expert that creates and maintains the Prolog control code for the resources. The function of the Code Expert is to reduce the developer's workload by allowing the developer to create a new application in only a few minutes and incrementally enhance this from a prototype to the final application.
- It has a language sensitive text-editor. The source code editor comes with colour coding of keywords and other language elements. This makes it much easier to differentiate between names, parameters, comments and so on.
- Possesses a powerful mechanism for handling error situations.
- Has a debugger with the ability to set breakpoints and step through coding. Inspection on variable values and on the contents of the asserted facts is performed during this process.
- Visual Prolog has a very fast, powerful, and flexible database subsystem, which makes it a suitable choice for database applications.
- It happens to be a powerful platform for building Client-Server applications.
- It also has portable SQL bindings that allow it to link programs to external databases when required.

2.5 Conclusion

Based on the findings and having done the analysis, the author has a better idea of what is expected of his system. The three systems that were reviewed have given the author a better understanding of what rule-based expert systems are. As for the development tool, the author has decided to use Visual Prolog because of its advantages over WIN-PROLOG.

Chapter 3

Methodology

3.0 Introduction

In this chapter, the system development methodology is discussed in detail. Besides that, a comprehensive study on rule-based reasoning and its concepts and functions will be carried out.

3.1 System Development Methodology

The development of a system requires a suitable development methodology. A methodology consists of several processes: requirements analysis, system design, module design, coding, unit testing, integration testing and system testing. The implementation of this methodology is important in ensuring a successful development process. The system development methodology in Figure 3.0 describes the aviation expert system's development process.

The methodology used is a combination of the waterfall model and the incremental prototyping model [9]. The waterfall model presents a high-level view of what will go on during the development of the system and the sequence of events that are to be encountered. As systems evolve, the problems that plague them become comprehensible and the alternatives to overcome or avoid these problems are evaluated. Therefore, the developer would have to back track to the previous phase to make enhancements. Backtracking through the entire waterfall model is not feasible, as it would require an ample amount of time and effort.

The main drawback of the waterfall model is that it fails to reflect the way systems are really developed. The author has decided to integrate the incremental prototyping model into the development methodology. This would help overcome the disadvantages and weaknesses of the waterfall model. The incremental prototyping model, in this case, is a more efficient and flexible way to develop the system. It enables the author to assess alternative system designs and coding strategies and decide which would be best suited for the development of the system. Revisions can be made at early stages, rather than at the end of the development cycle, which would require backtracking to the earlier stages of development.

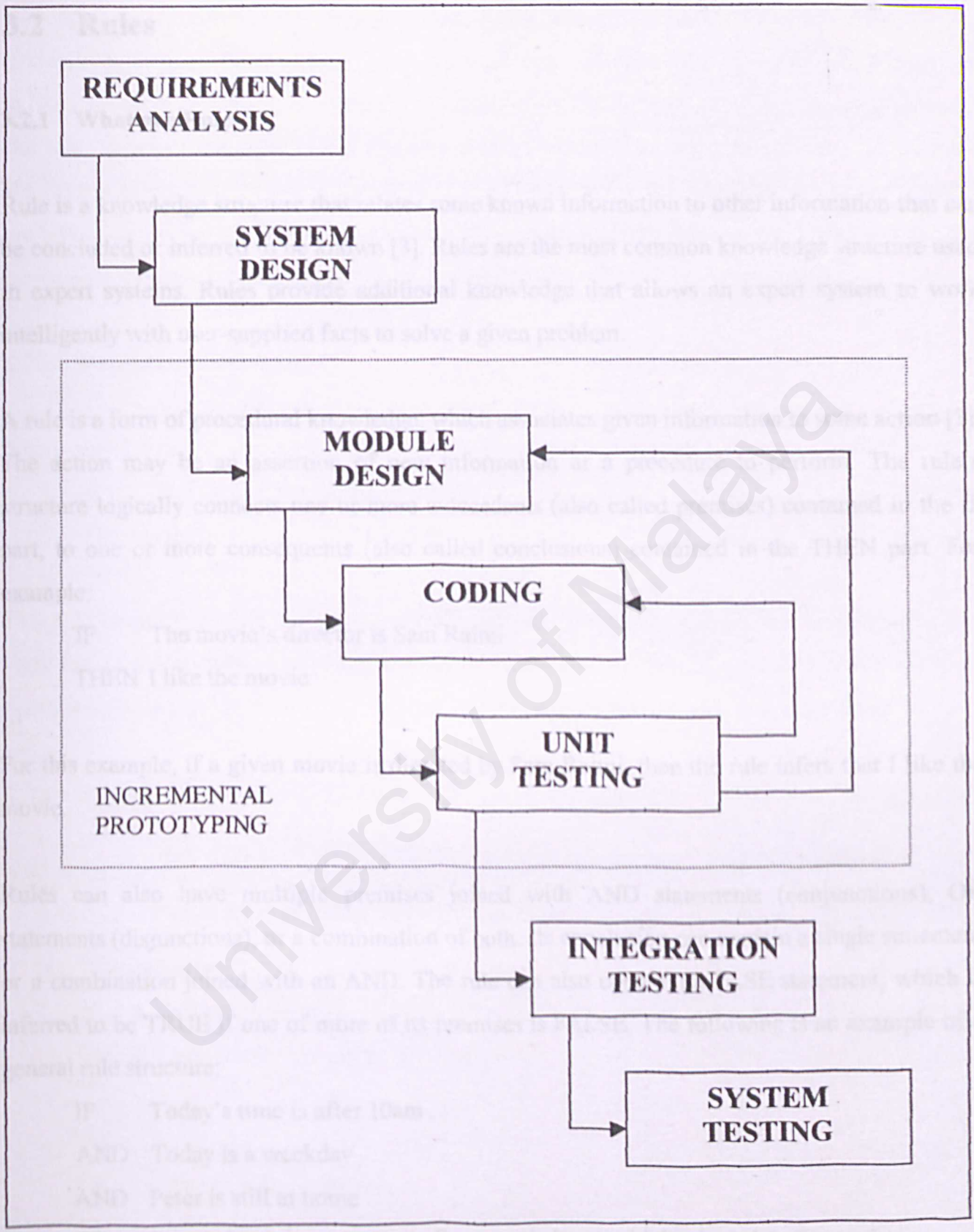


Figure 3.0 System Development Methodology

3.2 Rules

3.2.1 What are Rules?

Rule is a knowledge structure that relates some known information to other information that can be concluded or inferred to be known [3]. Rules are the most common knowledge structure used in expert systems. Rules provide additional knowledge that allows an expert system to work intelligently with user-supplied facts to solve a given problem.

A rule is a form of procedural knowledge, which associates given information to some action [3]. The action may be an assertion of new information or a procedure to perform. The rule's structure logically connects one or more antecedents (also called premises) contained in the IF part, to one or more consequents (also called conclusions) contained in the THEN part. For example:

IF The movie's director is Sam Raimi
THEN I like the movie

For this example, if a given movie is directed by Sam Raimi, then the rule infers that I like the movie.

Rules can also have multiple premises joined with AND statements (conjunctions), OR statements (disjunctions), or a combination of both. Its conclusion can contain a single statement or a combination joined with an AND. The rule can also contain an ELSE statement, which is inferred to be TRUE if one of more of its premises is FALSE. The following is an example of a general rule structure:

IF Today's time is after 10am
AND Today is a weekday
AND Peter is still at home
OR Peter's boss from the Daily Bugle called and said that he is late for work
THEN Peter is late for work
ELSE Peter is not late for work

In a rule-based expert system, domain knowledge is captured in a set of rules and entered in the system's knowledge base. These rules are used with information in the working memory to solve a problem. When the IF part of the rule matches the information in the working memory, the rule 'fires', causing the THEN statements to be added to the working memory. This causes the system to perform the action specified in the THEN part of the rule. The new statements that are added to the working memory can also start a chain reaction causing other rules to fire. The processing of rules in a rule-based expert system is managed by the inference engine.

3.2.2 Executing a Procedure

Rules can also perform operations like simple calculations as shown in the example below:

IF The area of the square is needed
THEN AREA = LENGTH*WIDTH

This rule would fire when the information shown in its premise is added to the working memory, resulting in the computation of the area. Rule-based systems can also perform more complex operations by designing it to access external programs such as databases or spreadsheets.

3.2.3 Types of Rules

Rules represent various forms of knowledge:

- **Relationship**

  IF The battery is dead
  THEN The car will not work

- **Recommendation**

  IF The car will not work
  THEN Take a cab

- **Directive**

  IF The car will not start

  AND The fuel system is ok

  THEN Check out the electrical system

- **Strategy**

  IF The car will not start

  THEN First check out the fuel system then check out the electrical system

- **Heuristic**

  IF The car will not start

  AND The car is a 1957 Ford

  THEN Check the float

3.2.4 Variables Rules

In some applications, where the same operation must be performed on a set of similar objects, writing a single rule for each object makes the system inefficient and hard to maintain. Consider the rule that concludes whether Bruce Wayne can retire:

IF Bruce Wayne is an employee

AND Bruce Wayne's age is greater than 65

THEN Bruce Wayne can retire

The rule checks if Bruce Wayne is an employee of the company and whether he is over the age of 65 before concluding he can retire. Writing separate rules for each individual is an inefficient process. Furthermore, if the retirement age changes, changes must be made to every single rule. Using variables, the rule would be as follows:

IF ?X is EMPLOYEE

AND ?X AGE > 65

THEN ?X can retire

This rules scans the working memory looking for matches to the two premises. If matches are found, then the rule concludes that the person can retire. These are called pattern-matching rules. They offer an efficient way to process information, which eases system coding and maintenance.

3.2.5 Uncertain Rules

One of the characteristics of an expert system is that it permits inexact reasoning. The domain expert will often provide a rule that establishes an inexact association between the premise and conclusion:

IF The grossing is high

THEN *Almost certainly* there will be a sequel

The domain expert believes that if the grossing is high, almost certainly there will be a sequel. Certainty factors (CF) are used to capture the confidence of the expert. Certainty factor is a numeric value assigned to a statement that represents the degree of belief in the statement [3]. Using certainty factors, the rule would be as follows:

IF The grossing is high

THEN There will be a sequel CF=0.8

3.2.6 Meta-Rules

Usually, the expert's knowledge is a solution to a problem. Sometimes, an expert uses knowledge that directs the problem solving process. This knowledge determines the best way to solve a problem using the existing domain knowledge. This type of knowledge is called meta-knowledge. Meta-knowledge is defined as knowledge about the use and control of domain knowledge [3].

Like any other knowledge, meta-knowledge can be represented in rules. Meta-knowledge is usually represented in meta-rules. Meta-rule is a rule that describes how other rules should be used [3]. A meta-rule devises a strategy on how to use domain-specific rules instead of concluding new information. The following example clearly states this:

IF The car will not start

AND The electrical system is operating normally

THEN Use the rules concerning the fuel system

3.2.7 Rule Sets

Rules alone are not sufficient for expert reasoning. Strategies are needed to know when and how to apply them. Through experience, an expert forms several sets of rules, each of which apply to different types of problems. With rule sets, specific rules are used only when appropriate. This eases the system's development and maintenance, allowing developers to focus their attention on one module at a time. Building, testing, and later modifying each module be done individually.

3.3 Reasoning

3.3.1 Introduction

Reasoning is the process of working with knowledge, facts, and problem solving strategies to draw conclusions [3]. There are a few types of reasoning:

- Deductive reasoning
- Inductive reasoning
- Abductive reasoning
- Analogical reasoning
- Common-sense reasoning
- Non-monotonic reasoning

3.3.2 Deductive Reasoning

Humans use deductive reasoning to deduce new information from logically related known information [3]. Deductive reasoning uses problem facts or axioms and related general knowledge in the form of rules or implications. The axioms are compared with a set of implications to conclude new axioms. For example:

Implication: I will get the ticket if I am early

Axiom: I am early

Conclusion: I will get the ticket

Deductive reasoning is logically pleasing and is a very common human problem-solving technique. The modus pones rule of inference is the basic form of deductive reasoning:

If A is true and if A implies B is true, then B is true.

3.3.3 Inductive Reasoning

Humans use inductive reasoning to arrive at a general conclusion from a limited set of facts by the process of generalization [3]. For example:

Premise: The movie grossed a lot of money in America

Premise: The movie grossed a lot of money in Asia

Conclusion: In general, the movie grossed a lot of money everywhere

In inductive reasoning, a generalization, which applies to all cases, are formed from the knowledge that applies to a limited number of cases. The induction process is described as:

For a set of objects, X={a, b, c, d...}, If property P is true for a, and if P is true for b, and if P is true for c... then P is true for all X.

3.3.4 Abductive Reasoning

Abduction is a form of deduction that allows for plausible inference. This means that the conclusion may follow from the available information, but it might be wrong. For example:

Implication: Clothes are dry if the sun is shining

Axiom: Clothes are dry

Conclusion: Is the sun shining?

The inference is said to be plausible because the clothes could be dry because of other reasons, such as a strong wind. The abduction process is described as:

If B is true and if A implies B is true, then A is true?

3.3.5 Analogical Reasoning

Through experiences, humans form a mental mode of some concept to help them understand a situation or object. This is done through analogical reasoning. A comparison is done between the model and situation or object, to find similarities or differences to guide their reasoning. Consider the following frame to better understand what analogical reasoning is:

LION Frame

Specialization-of: ANIMALS

Number-of-legs: 4

Eats: meat

Lives: Africa and India

Colour: tawny

In this case, the frame is the mental model, which helps to understand a situation or object. If it was said that a tiger is like a lion, it is then assumed that they share many features. The differences between the mental model and the situation or object help enhance the understanding of the situation or object.

3.3.6 Common-Sense Reasoning

Common sense reasoning relies more on good judgments than on exact logic [3]. For example, the consider the statement:

A loose fan belt usually causes strange noises

A mechanic might have formed this common sense knowledge from his experiences with working on many automobiles. This type of knowledge is referred to as heuristics - rule of thumb. When heuristics are used to guide the problem solving in an expert system, it is called heuristic search or best-first search. This type of search looks for solutions in the most likely places. It doesn't provide a guarantee that a solution will be found in the direction taken, only that it s a reasonable one. Heuristics searches usually provide faster solutions, because the path taken skips unnecessary steps.

3.3.7 Non-Monotonic Reasoning

Monotonic reasoning is where a problem is solved using information that is static during the problem-solving period. In some problems, the state of facts can change. Adjusting dependent information according to these changes is known as non-monotonic reasoning. An expert system can perform non-monotonic reasoning if it has a truth maintenance system. A truth maintenance system keeps track of what caused a fact to be asserted. If the cause is removed, then the fact is also removed.

3.4 Inference

3.4.1 Introduction

Expert systems model the reasoning process of humans using a technique called inference. Inference is the process used in an expert system to derive new information from known information. The inference process is performed using the inference engine.

3.4.2 Modus Ponens

Modus Ponens is a rule of logic that asserts that if we know A is true and that A implies B is true, then we can assume that B is true [3].

IF A is true

AND A → B is true

THEN B is true

Modus Ponens works with axioms (truth statements) to infer new facts. For example, if there is an axiom of the form $E^1 \rightarrow E^2$ and there is another one of the form E^1, then E^2 is logically inferred to be true. The axioms can be compiled into a list where axiom 3 follows from 1 and 2:

1. E^1
2. $E^1 \rightarrow E^2$
3. E^2

Modus Ponens forms a series of logical assertions from working with a set of implications (rules) and initial data. This type of inference process is driven by asserted information, which is the basis of forward-chaining expert systems.

3.4.3 Resolution

Modus Ponens forms a series of logical assertions from working with a set of initial data, where it is important to learn as much as possible form available information. But in some applications, we need to gather specific information to prove some goal. This type of inference is the basis of resolution, which is the basic algorithm used in Prolog. Resolution is defined as an inference strategy used in logical systems to determine the truth of an assertion [3].

The resolution method attempts to prove that a goal, expressed as proposition P is TRUE, given a set of axioms about the problem. This method actually attempts to prove that –P cannot be TRUE, and involves the producing of new resolvents from the union of existing axioms and the negated goal. For example:

There exist two axioms: $A \vee B$ (A is true or B is true)

$-B \vee C$ (B is not true or C is true)

The resolution method forms the resolvent of these two axioms by joining them with a logical AND:

$$(A \vee B) \wedge (-B \vee C) = A \vee C$$

This result is obtained through logical addition:

$$\begin{array}{l} A \vee B \\ \underline{-B \vee C} \end{array}$$

giving $A \vee C$ since B and -B cancel

These resolvents are added to the list of axioms and new resolvents are derived. The process continues until a contradiction (two axioms that are logically contradictory, such as P and –P) is produced.

3.5 Forward-Chaining

The solution process for most problems begins by collecting information about the problem. The information collected is then reasoned with, to infer logical conclusions. This style of reasoning, called forward-chaining, is modeled in an expert system using a data-driven search.

Forward-chaining is defined as an inference strategy that begins with a set of known facts, derives new facts using rules whose premises match the known facts, and continues this process until a goal state is reached or until no further rules have premises that match the known or derived facts [3].

These are steps of a simple application of forward-chaining in a rule-based expert system:

1. The system obtains problem information from the user and places it in the working memory.
2. The inference engines scans the rules looking for one whose premises match the contents in the working memory.
3. If a rule is found, the system adds this rule's conclusion to the working memory (firing the rule) and then cycles and checks the rules again looking for new matches.
4. On the new cycle, previously fired rules are ignored.
5. The process continues until no matches are found.

3.6 Rule-Based Systems

3.6.1 Introduction

A rule-based expert system is a computer program that processes problem-specific information contained in the working memory with a set of rules contained in the knowledge base, using an inference engine to infer new information [3].

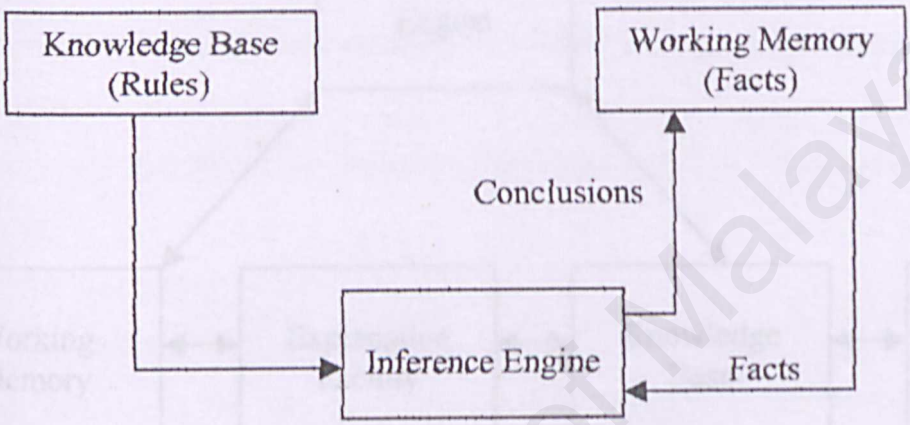


Figure 3.1 Rule-Based Model

Knowledge base: models a human's long-term memory as a set of rules

Working memory: models a human's short-term memory and contains problem facts both entered and inferred by the firing of the rules

Inference engine: models human reasoning by combining problem facts contained in the working memory with rules contained in the knowledge base to infer information

3.6.2 Rule-Based System Architecture

The three components mentioned earlier are the most important parts of a rule-based expert system but there are other subsystems that can be found in any real system.

User interface: the vehicle through which the user views and interacts with the system

Developer interface: the vehicle through which the knowledge engineer develops the system

Explanation facility: the subsystem responsible for providing explanations on the reasoning of the system

External programs: programs such as databases, spreadsheets, algorithms, etc, that work in a supporting role for the system

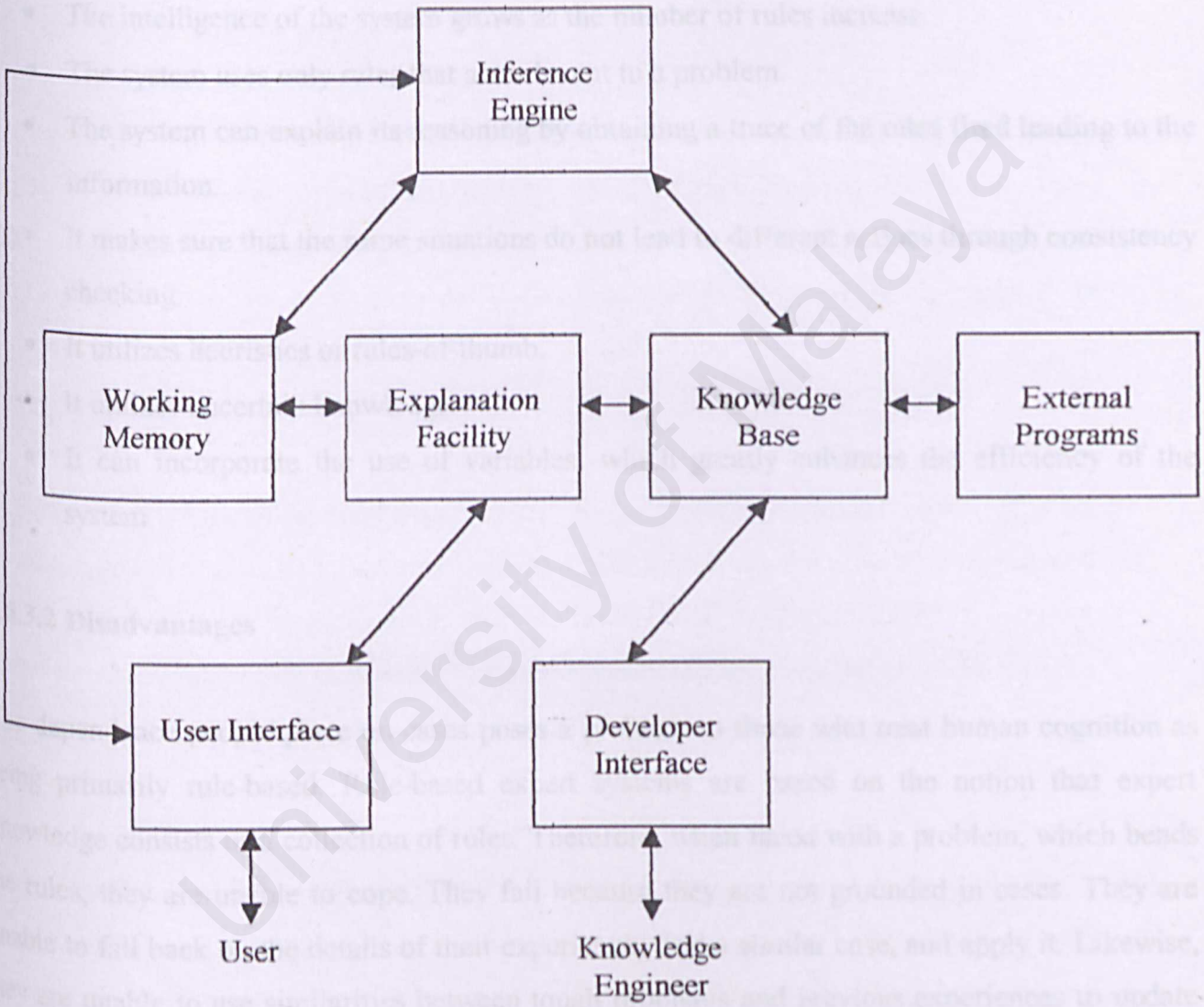


Figure 3.2 Rule-Based System Architecture

3.6.3 Advantages and Disadvantages of Rule-Based Systems

3.6.3.1 Advantages

- It can capture natural human problem-solving knowledge.
- Its separation of control from its knowledge eases system modification and maintenance.
- Its separation of control from its knowledge allows the system to be expanded easily.
- The intelligence of the system grows as the number of rules increase.
- The system uses only rules that are relevant to a problem.
- The system can explain its reasoning by obtaining a trace of the rules fired leading to the information.
- It makes sure that the same situations do not lead to different actions through consistency checking.
- It utilizes heuristics or rules-of-thumb.
- It utilizes uncertain knowledge.
- It can incorporate the use of variables, which greatly enhances the efficiency of the system.

3.6.3.2 Disadvantages

The dependence people place on cases poses a problem to those who treat human cognition as being primarily rule-based. Rule-based expert systems are based on the notion that expert knowledge consists of a collection of rules. Therefore, when faced with a problem, which bends the rules, they are unable to cope. They fail because they are not grounded in cases. They are unable to fall back on the details of their experience, find a similar case, and apply it. Likewise, they are unable to use similarities between tough problems and previous experiences to update their rules. Their failure to retain cases cripples their ability to learn from their experiences.

3.7 The flex Toolkit

In this section we take a look at one of the many expert system toolkits available in the market. The flex expert system toolkit is an expressive and flexible AI toolkit, available on a wide range of hardware and software platforms. It provides a comprehensive and versatile set of facilities for programmers to construct sophisticated, readable and portable expert systems [10].

A Hybrid Expert System Shell

The flex toolkit is an expressive and powerful expert system toolkit, which supports frame-based reasoning with inheritance, rule-based programming and data-driven procedures fully integrated within a logic programming environment. To make these constructs accessible in an intuitive way, flex contains its own dedicated English-like Knowledge Specification Language (KSL).

A Dynamic Toolkit

The flex toolkit goes beyond most expert system shells in that it employs an open architecture and allows you to access, augment and modify its behavior through a layer of access functions. Because of this, flex is often referred to as an AI toolkit. The combination of flex and Prolog results in a functionally rich and versatile expert system development environment where developers can fine tune and enhance the built-in behavior mechanisms to suit their own specific requirements.

The flex toolkit appeals to various groups of developers; expert systems developers who want to deliver readable and maintainable knowledge-bases, advanced expert system builders who want to incorporate their own controls, AI programmers who want access to a high-level language-based product and Prolog programmers who require extra functionality.

The flex Toolkit Structures

The main structures in flex are frames and instances with slots for organizing objects, default and current values for storing data, demons and constraints for adding functionality to slot values, rules and relations for expressing knowledge and expertise, functions and actions for defining imperative processes, and questions and answers for end-user interaction.

Knowledge Specification Language

The flex toolkit has its own expressive English-like Knowledge Specification Language (KSL) for defining rules, frames and procedures. The KSL enables developers to write simple and concise statements about the expert's world and produce virtually self-documenting knowledge bases, which can be understood and maintained by non-programmers. The KSL supports mathematical, Boolean and conditional expressions and functions along with set abstractions. Furthermore, the KSL is extendable through synonyms and templates. By supporting both logical and global variables in rules, flex avoids unnecessary rule duplication and requires fewer rules than most other expert systems.

Chapter 4

Requirements Analysis And System Design

4.0 Introduction

This chapter discusses the system analysis and design, which includes a study on the functional and non-functional requirements, as well as software and hardware requirements for developing the proposed system and the system design.

4.1 Requirements Analysis

Requirements analysis is the process of acquiring the needs of a system based on the user's preferences. The purpose of this process is to develop an understanding of the problem domain and the specification of the system [9]. This process can be divided into two parts, which are functional requirements and non-functional requirements. Functional requirements refer to the interaction between the system and its environment, what the system will do and how the system should behave. In other words, it outlines the system flow, which explains how the system will interact with the user. Non-functional requirements on the other hand, define the constraints of the system, such as robustness, reliability, modularity, consistency, and user-friendliness.

4.1.1 Functional Requirements

Functional requirements are an interaction between the system and its environment [11]. Functional requirements give an outline on how the system should behave when in use. How the system runs from screen to screen is determined in this section. Although the description of the system flow may not be very specific, the system will be designed based on these requirements. For example, the user has to make a selection from available options on one screen in order to proceed to the next screen. If this is not done the system will prompt the user to do so. These are identified functional requirements for the proposed aviation expert system:

- Read Input From User
- Match Input With Rules
- Recommend Solution(s)

4.1.1.1 Read Input From User

When the user logs in to the system, there will be several options displayed for the user to choose from. The user has to make a selection in order to proceed to the next screen. The choices made by the user are information that will be placed in the working memory. In the aviation expert system, the user will first be asked to choose the type of failure, whether it is a single or multiple failure. The next screen will prompt the user to choose which system is giving trouble. If it is a multiple failure situation, the user will have to choose all the systems involved. Finally, the user will have to choose the alert messages displayed.

4.1.1.2 Match Input With Rules

The information provided by the user would be evaluated using rule-based reasoning techniques. This evaluation is performed by the inference engine, which will scan the rules in the knowledge base looking for one whose premises matches the information in the working memory. If a rule is found, the system adds this rule's conclusion to the working memory and then cycles and checks the rules again looking for new matches. This process is repeated until no matches are found.

4.1.1.3 Recommend Solution(s)

Once the matching process has come to a halt, all the conclusions in the working memory will be displayed. In some cases, there may only be a single solution to be displayed. In the aviation expert system, the solution(s) displayed will be a counter measure that must be taken to overcome the problem(s) at hand.

4.1.1.4 System Flow

Figure 4.0 on the next page illustrates the system flow discussed in the previous section.

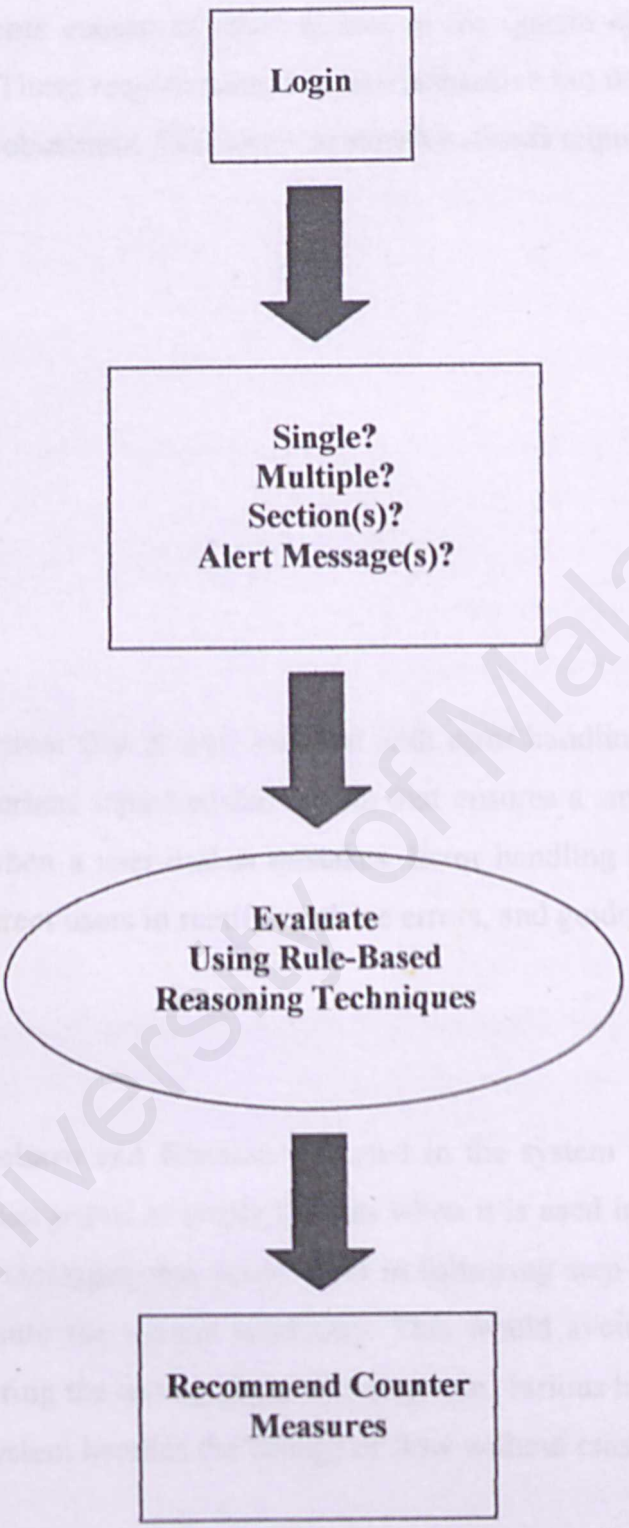


Figure 4.0 System Flow

4.1.2 Non-Functional Requirements

Non-functional requirements consist of other factors in the system development that must be taken into consideration. These requirements are very subjective but they play an important role when it comes to system robustness. These are the non-functional requirements identified for the proposed system:

- Robustness
- Reliability
- Modularity
- Consistency
- User Friendliness

4.1.2.1 Robustness

Robustness refers to a system that is well rounded with error handling and validation features. Error handling is an important aspect of the system that ensures a smooth system flow. It will prompt error messages when a user makes mistakes. Error handling features would be able to display error messages, direct users in rectifying those errors, and guide them accordingly.

4.1.2.2 Reliability

A reliable system will behave and function as stated in the system specification. The system should not produce any dangerous or costly failures when it is used in the manner stated in the systems manual. Pop-up messages that guide users in following step-by-step procedures would be a good feature to ensure the system reliability. This would avoid any unwanted errors or failures during usage. During the testing phase of the system, various handling patterns should be tested out to ensure the system handles the change of flow without crashing.

4.1.2.3 Modularity

Systems should be developed using a modular approach to enable the developer to easily make enhancements in the future. If there were changes in functionality or scope, modifications can be made easily. The ability to do this is attained if the program is written in modular. It makes the program easier to understand, which in turn makes it easier to amend after a long period of time. It also allows reusability of certain functions and common procedures.

4.1.2.4 Consistency

Certain physical features such as screens or windows should have the same design to ensure consistency of the system. Commands like Exit, Main Menu, and Reset would be displayed on every screen for consistency. The availability of these commands on every screen would enable users to exit the program, reset the options made, or to start all over again if there was a mistake in entry while using the system. This feature also indirectly enhances simplicity of the program for new users who are not proficient enough with the system.

4.1.2.5 User Friendliness

User interface is an important feature in determining the usability of a system. The proposed system will be designed with an interactive user interface that provides a simple and straightforward approach for users. This is because a sophisticated or attractive design may cause the user interface to be unnecessarily cluttered with fancy content thus taking the attention away from the more important aspects of the system. A good user interface has a balance of attractiveness and accessible features while maintaining professionalism in its design.

The use of suitable icons and buttons will make it easier for users to navigate the system. The system will also deploy the use of a pop up message, which requests confirmation when users exit the system. The system will also have appropriate prompts and instructions to guide the user through the operation of the system.

4.2 Software Requirements

These are the list of software that will be used to develop the system:

- Windows 98/XP
- Visual Prolog 5.2
- Microsoft Word 2000 (documentation purposes)

4.3 Hardware Requirements

These are the list of hardware requirements that will be used to develop the system:

- Pentium III-MMX at 450 MHz
- 512 KB Pipeline Burst Cache
- 192 MB RAM
- 20 GB Hard Disk
- SONY CD-RW CRX0811
- Other standard desktop PC components

These are not specifications of minimum hardware requirements to develop the system. Rather, it is the hardware configuration of the author's personal computer, which will be used to develop the system.

4.4 System Design

Design is a creative and iterative problem-solving process. The goal of this phase is to develop a model of the proposed system based on the set of requirements defined in the previous section. It is difficult to determine the best design that may suit the proposed system. However, a good design should have the following characteristics:

- Ease of understanding
- Ease of implementation
- Ease of testing
- Ease of modification
- Correct translation of what is specified in the requirement analysis

The system architecture design of the Aviation Expert System is based on:

- Literature survey done on other expert systems.
- The methodology of the system.
- User requirements
- The overall objective and scope of the system

Figure 4.1 on the next page illustrates this system design.

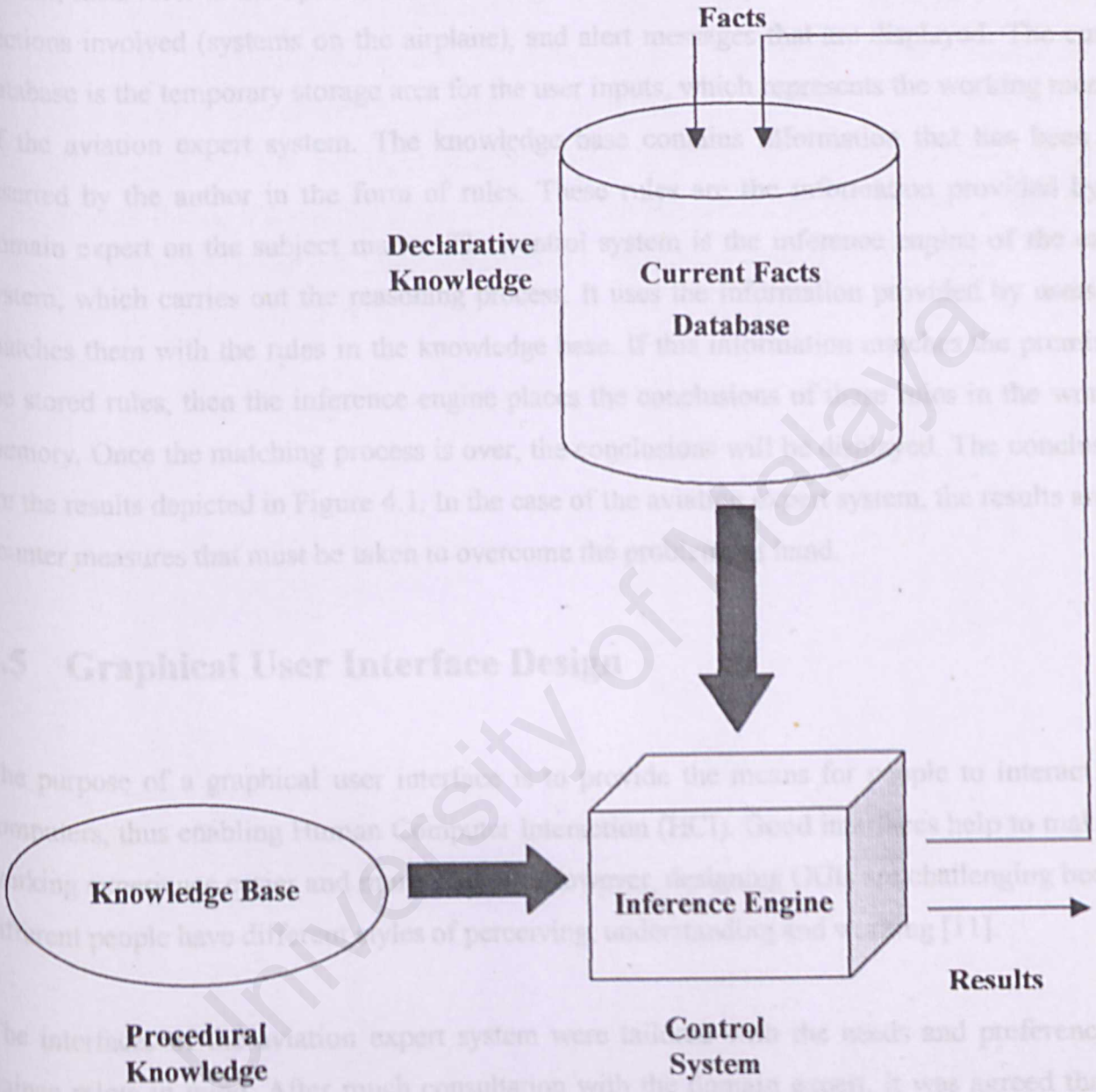


Figure 4.1 System Architecture Design

Figure 4.1 is actually a basic system architecture design of an expert system. However, the author will relate the proposed aviation expert system to this design. Facts are information that are inserted by users and will be used in the reasoning process. In the case of the aviation expert system, facts refer to the options the user chooses, which are type of failure (single or multiple), sections involved (systems on the airplane), and alert messages that are displayed. The current database is the temporary storage area for the user inputs, which represents the working memory of the aviation expert system. The knowledge base contains information that has been pre-inserted by the author in the form of rules. These rules are the information provided by the domain expert on the subject matter. The control system is the inference engine of the expert system, which carries out the reasoning process. It uses the information provided by users and matches them with the rules in the knowledge base. If this information matches the premise of the stored rules, then the inference engine places the conclusions of these rules in the working memory. Once the matching process is over, the conclusions will be displayed. The conclusions are the results depicted in Figure 4.1. In the case of the aviation expert system, the results are the counter measures that must be taken to overcome the problems at hand.

4.5 Graphical User Interface Design

The purpose of a graphical user interface is to provide the means for people to interact with computers, thus enabling Human Computer Interaction (HCI). Good interfaces help to make the working experience easier and more pleasant. However, designing GUIs are challenging because different people have different styles of perceiving, understanding and working [11].

The interfaces of the aviation expert system were tailored with the needs and preferences of trainee pilots in mind. After much consultation with the domain expert, it was agreed that the GUIs should have minimal graphics and presented the alert messages and counter measures clearly.

For a presentation and explanation of the GUIs, please refer to the User Manual in Appendix A.

4.6 Conclusion

This chapter covered the requirements analysis and the system design. The requirements analysis included functional and non-functional analysis, which branched out to software and hardware requirements for developing the system. The system architecture design of the proposed system was also laid out in this chapter. As previously mentioned in Chapter 3, a combination of the waterfall and incremental prototyping methodology is used to ensure flexibility of future enhancements during the coding phase. Thus, the system design in this chapter is the system architecture to be implemented, but it is subjected to changes as new ideas may arise.

Chapter 5

System Implementation

Managing & Creating Images

- Adobe Photoshop 5.0

Adobe Photoshop 5.0 was used for the purpose of creating and manipulating images. Besides that, it was also essential in converting all JPEG and GIF images into Bitmaps as Visual Prolog only supports images in this form.

5.2 Program Coding

The phases involved in program coding are described in brief below.

Prototype Development

In the first phase, the author designed a simple prototype that used rule-based reasoning as a problem solving methodology. This prototype contained only a few alert messages and conditions for the Un-annunciated Checklist. The output of this prototype (counter measures) was text-based, with no user interface.

User Interface Development

This phase involved the designing of an appropriate interface for the prototype. This was done with the Application Expert and the Dialog Package Expert. Visual Prolog is similar to Visual Basic when it comes to designing interfaces. They both share the same interface development characteristics, which utilises the 'drag and drop' feature. Therefore, a simple and user-friendly interface was developed in no time.

Linking the Prototype to the Interface

Visual Prolog's Code Expert helps to generate codes for relevant controls, along with documentation that allows it to track clauses used for the specific control. Controls in the

interface, such as push buttons and radio buttons were linked to the prototype with the help of the Code Expert.

Expanding the Scope

Now that the author had a working system that was able to recommend counter measures, it was time to expand the scope to the other alert messages and conditions in the Un-annunciated Checklist and from there, further expand the system to the other 15 parts and systems of the airplane. The parts and systems of an airplane were studied thoroughly to build this extremely detailed expert system. The system would not only provide counter measures to every alert message and condition, but also an explanation on why certain counter measures are taken. The end result is an expert system that features more than 650 dialogs and over 50, 000 lines of coding.

Final Touch Ups

The last phase of the system implementation involved activities carried out to make the system user-friendlier. Screen modifications such as adding images and repositioning of controls were done to ensure an uncomplicated experience when browsing through the system. This aspect is crucial as the aviation expert system is suppose to prepare trainee pilots for critical situations where lives are at stake. Therefore, after consultation with the domain expert, the author ruled out creating windows that were deemed too fancy.

5.2.1 GUI Implementation

The GUI was designed based on the user requirements mentioned in Chapter 4 of this thesis. Visual Prolog's drag-and-drop feature made it easy to design the user interface. The aviation expert system consists fully of dialogs. Unlike most applications developed in Visual Prolog, the aviation expert system does not use the Task Window, which is usually used if the application developed involves various types of functionality, such as linking the program to an operating

system. Therefore, the author chose to develop the aviation expert system solely based on dialogs. The system was designed using the Window and Dialog Editor shown below.

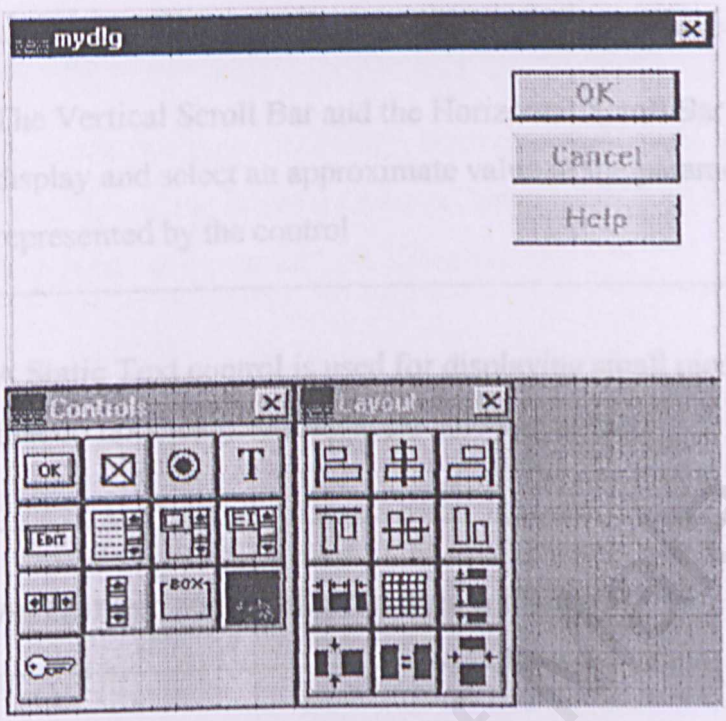


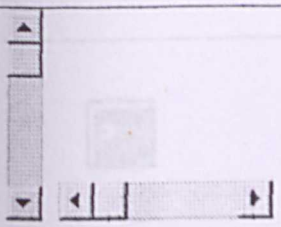
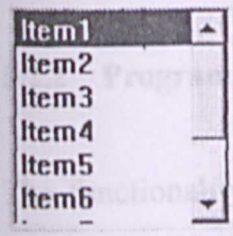
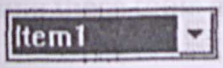
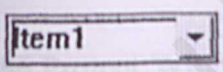
Figure 5.0 The Window and Dialog Editor

The Window and Dialog Editor comes with two special toolbars that contain attributes for controls and layouts. Table 5.0 describes the various types of controls.

Types of Controls

Table 5.0 Types of Controls

Control	Description
Push Button	A Push Button executes code that implements the action or function of the button
Check Box	A Check Box is a two state button, which is used to indicate a Boolean state

Radio Button	Radio Buttons are used in groups to indicate one choice from several possibilities
	The Vertical Scroll Bar and the Horizontal Scroll Bar are used to display and select an approximate value of the parameter represented by the control
Static text	A Static Text control is used for displaying small pieces of text that do not change during application execution
Edit	An Edit text control allows a user to input text
	A List Box contains a scrollable list of elements for a user to choose form
	A List Button is used to select one item from a set. The List Box control is a combination of the Edit Box and List Box controls
	A List Edit control is a combination of the Edit and List Box controls. The function is similar to the Edit control but the ↓ button allows a list of predefined strings to pop up for further editing
GroupBox	A Group Box is used to indicate visually that the set of controls within it has some meaning as a group. It has no functional effect

	An Icon control allows you to display in a window or a dialog an Icon defined in the resources. They are small bitmaps that are mapped to the pixels of the display
	There are two types of Custom controls – the External Custom Controls and the Internal Custom Controls. Custom Controls are distributed as Dynamic Link Libraries

Inserting the Controls

These steps must be followed to insert a control into a dialog.

- First, click a control from the Controls toolbar.
- Then, move the cursor to a position in the dialog where you want the control placed.
- Click, and edit the control attributes.

5.2.2 Programming

The functionality of the aviation expert system was programmed using the Project Window, Dialog and Window Expert and the Dialog Pack Expert. A sample of the coding that was done using these tools are explained in detail in Appendix C of this thesis.

Project Window

The Project Window appears when a project is opened. It lists all the components of a Visual Prolog application by type. The list box in the middle of the Project Window shows all the components of the corresponding component type buttons on the left. The buttons on the right side of the Project Window activates the tools you need to edit a component. Double-clicking the component activates the Window and Dialog Editor for that particular component.

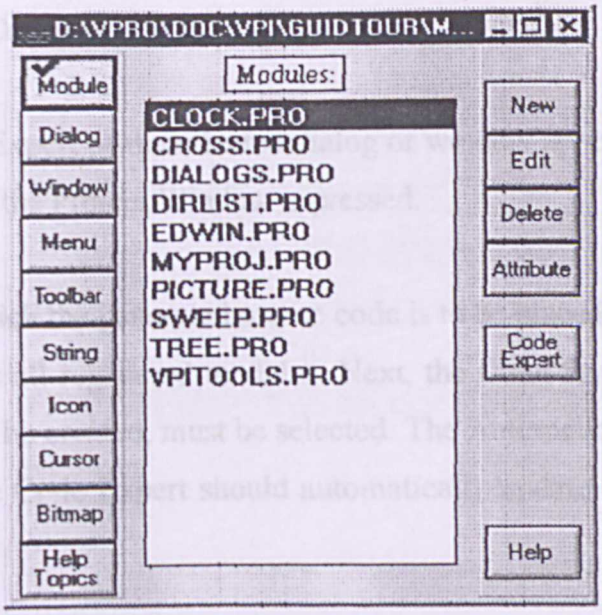


Figure 5.1 The Project Window

Dialog and Window Expert

The Dialog and Window Expert connects Prolog codes to the layout of windows and dialogs. It inserts the necessary codes needed to manage window and dialog creation and event handling.

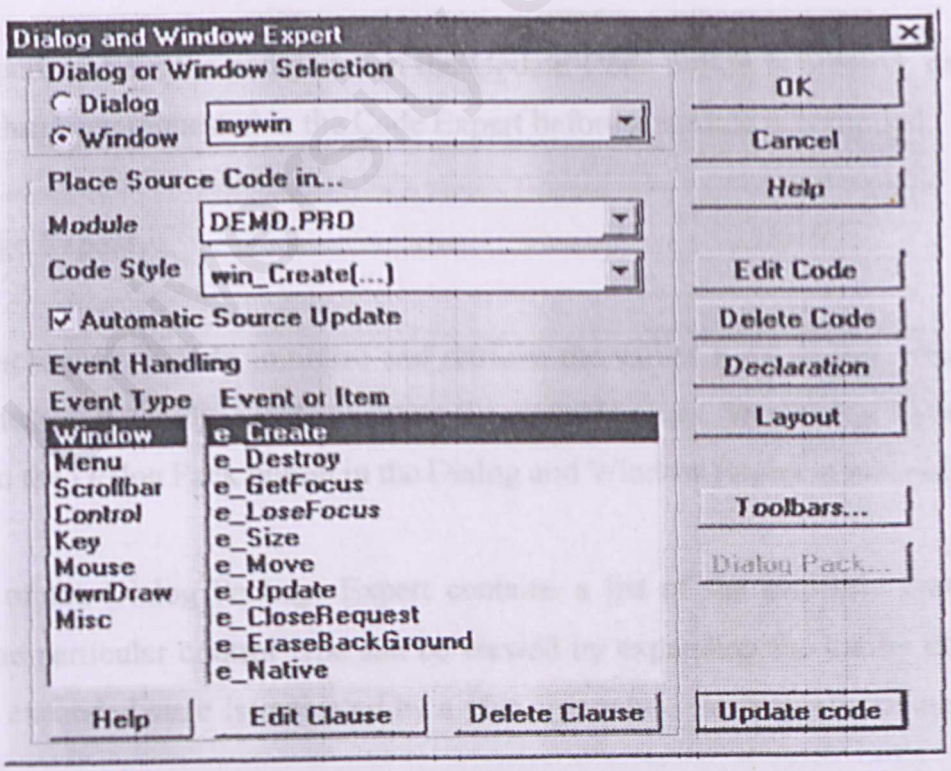


Figure 5.2 The Dialog and Window Expert

Generating the Default Code for a Window

The Dialog and Window Expert of the selected dialog or window appears when the Code Expert button on the right side of the Project Window is pressed.

The Prolog Module, in which the generated source code is to be placed in, must be selected using the list button, which lists all registered modules. Next, the Code Style, which determines how the dialog or window will be created, must be selected. The Automatic Source Update checkbox selects whether or not the Code Expert should automatically update the source code whenever the layout changes.

When the above fields have been selected, the Default Code button will be activated. If this button is grayed, an inappropriate code style might have been selected. Pressing the Default Code button will cause a global declaration of the predicate used for creating the window or dialog. This predicate will be added to the header of the *.pro file of the module while the creation predicate clauses and the event handling code will be added to the end of the file. The Default Code button will then change to the Update Code button.

The Code Expert updates the source when the Update Code button is pressed, or if automatic source update has been requested in the Code Expert before a module is compiled.

Dialog Package Expert

The Dialog Package is used to initialize and retrieve the values for a dialog, which includes a number of features for handling and validating the control values. The Dialog Package Expert is activated when the Dialog Pack button in the Dialog and Window Expert is pressed.

The left part of the Dialog Package Expert contains a list of the available controls. All the settings for one particular control type can be viewed by expanding the list by double-clicking the item. The expanded state is indicated by a plus sign while the original collapsed state by a minus sign.

The various settings for a selected item can be set on the right side of the Dialog Pack Expert. For certain controls, several values might be needed to initialize the control.

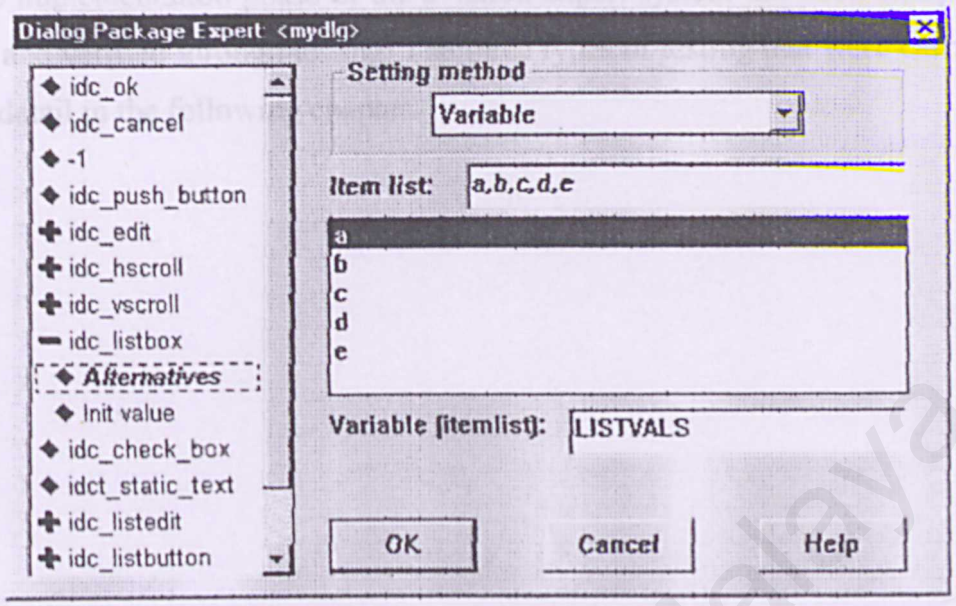


Figure 5.3 The Dialog Pack Expert

5.3 Conclusion

Now that the implementation phase of the aviation expert system has been completed, it has to be redefined and verified through testing. The three types of testing that were carried out will be discussed in detail in the following chapter.

Chapter 6

System Testing

6.0 Introduction

The three testing phases are unit testing, integration testing and system testing. The primary purpose of this phase is to examine the system for errors and rectify them. Since the aviation expert system determines the safety of passengers on board an aeroplane, it is important that the system is free of errors, as the slightest inaccuracy could prove to be fatal.

6.1 Unit Testing

Unit testing was performed throughout the development of the aviation expert system. Unit testing can be divided into three stages. First, the author examined the code to look for algorithm, data, and syntax errors. The code was compared with the specification and the system design to ensure all aspects of the proposed system were covered. Next the author compiled the code and eliminated all remaining syntax errors. Finally, test runs were done to make sure that an input produced the desired output.

6.1.1 Main Menu

The 'Main Menu' has three push buttons: 'Enter', 'Exit' and 'About'. These buttons were tested to see if the desired output was produced.

Case 1: 'Enter'
Result: 'Failure' screen is created

Case 2: 'Exit'
Result: System terminates

Case 3: 'About'
Result: 'About' screen is created

6.1.2 Failure

The 'Failure' screen has a set of two radio buttons and three push buttons. The set of radio buttons prompts the user to choose between 'Single' and 'Multiple' failures and the 'Enter' button proceeds to the pertaining screen. The other buttons are the '←' and 'Help' buttons.

Case 1: 'Single'
Result: 'Systems' screen is created

Case 2: 'Multiple'
Result: 'Test' screen is created

Case 3: '←'
Result: System recedes to 'Main Menu'

Case 4: 'Help'
Result: 'Help' screen for the 'Failure' screen is created

6.1.3 Systems

The 'Systems' screen has a set of 16 radio buttons and three push buttons. The set of radio buttons prompts the user to choose one of the 16 systems and the 'Enter' button proceeds to the pertaining screen. The 'Un-annunciated Checklist' is cited below as an example. The other buttons are the '←' and 'Help' buttons.

Case 1: 'Un-annunciated Checklist'
Result: The 'System' screen for 'Un-annunciated Checklist' is created

Case 2: '←'
Result: System recedes to the 'Failure' screen

Case 3: 'Help'

Result: 'Help' screen for the 'Systems' screen is created

6.1.4 System

The 'System' screen has a set of radio buttons and four push buttons. The set of radio buttons prompts the user to choose one of the many alert messages displayed and the 'Enter' button proceeds to the pertaining screen. The 'ABORTED ENGINE START' message of the 'Un-annunciated Checklist' is cited below as an example. The other buttons are the '←' and 'Help' and 'Exit' buttons.

Case 1: 'ABORTED ENGINE START'

Result: The 'Condition' screen for 'ABORTED ENGINE START' message is created

Case 2: '←'

Result: System recedes to the 'Systems' screen

Case 3: 'Help'

Result: 'Help' screen for the 'System' screen is created

Case 4: 'Exit'

Result: System terminates

6.1.5 Condition

The 'Condition' screen has a set of radio buttons and three push buttons. The set of radio buttons prompts the user to choose one of the many conditions encountered and the 'Enter' button proceeds to the pertaining screen. The 'Plane is on the ground' condition of the 'ABORTED ENGINE START' alert message is cited below as an example. The other buttons are the '←' and 'Help' buttons.

Case 1: 'Plane is on the ground'

Result: The 'Counter Measures' screen for 'Plane is on the ground' condition is created

Case 2: '←'

Result: System recedes to the 'System' screen

Case 3: 'Help'

Result: 'Help' screen for the 'Condition' screen is created

6.1.6 Counter Measures

The 'Counter Measures' screen has four or five push buttons: '#', 'Exit', 'Main Menu', '←', 'Help'. These buttons were tested to see if the desired output was produced.

Case 1: '#'

Result: The 'Reason' screen for a particular counter measure is created

Case 2: 'Exit'

Result: System terminates

Case 3: 'Main Menu'

Result: System returns to the 'Main Menu'

Case 4: '←'

Result: System recedes to the 'Condition' screen

Case 5: 'Help'

Result: 'Help' screen for the 'Counter Measures' screen is created

6.1.7 Reason

The 'Reason' screen has only one push button, which is the 'OK' button. This button was tested to see if the desired output was produced.

Case 1: 'OK'

Result: The 'Reason' screen terminates

6.1.8 Meaning

Sometimes, the 'System' screen does not proceed to the 'Condition' screen to prompt the user for the conditions encountered for the alert message selected. This is because the alert messages displayed by these systems are solely for information purposes and do not require counter measures. The 'Meaning' screen has only one push button, which is the 'OK' button. This button was tested to see if the desired output was produced. The 'ATC DATALINK LOST' message of the 'Communications' system is cited below as an example.

Case 1: 'ATC DATALINK LOST'

Result: The 'Meaning' screen for 'ATC DATALINK LOST' message is created

Case 2: 'OK'

Result: The 'Meaning' screen terminates

6.1.9 Test

The 'Test' screen has four push buttons: 'Test 1', 'Test 2', '←', and 'Help'. These buttons were tested to see if the desired output was produced.

Case 1: 'Test 1'

Result: The 'Test 1' screen is created

Case 2: 'Test 2'

Result: The 'Test 2' screen is created

Case 3: '←'

Result: System recedes to the 'Failure' screen

Case 4: 'Help'

Result: 'Help' screen for the 'Test' screen is created

6.1.10 Test X

The 'Test X' screen has three four buttons: 'Solutions', '←', 'Help' and 'Exit'. These buttons were tested to see if the desired output was produced.

Case 1: 'Solutions'

Result: The 'Solutions' screen is created

Case 2: '←'

Result: System recedes to the 'Test' screen

Case 3: 'Help'

Result: 'Help' screen for the 'Test X' screen is created

Case 4: 'Exit'

Result: System terminates

6.1.11 Solutions

The 'Solutions' screen has only one push button, which is the 'OK' button. This button was tested to see if the desired output was produced.

Case 1: 'OK'

Result: The 'Solutions' screen terminates

6.2 Integration Testing

Integration testing involves the testing of integrated modules or subsystems that have been incorporated into the system. The aviation expert system has two main modules, Single Failure and Multiple Failures. These two modules were integrated into one system and tests were done to see if they met the systems requirement and functions. The results showed that a successful integration of both modules had been achieved.

6.3 System Testing

System testing for the aviation expert system was done throughout the system development life cycle. The primary purpose for system testing is to ensure that the system requirements are met and the program functions as stated in the requirements analysis. Besides error detection, other aspects such as functionality testing also require a fair share of attention.

System testing can be divided into several other testing phases, such as functional testing, performance testing, acceptance testing, installation testing and additional testing.

6.3.1 Functional Testing

Functional testing ignores the system structure and focuses on the functionality of the program. This testing compares the systems actual performance with its requirements. For the aviation expert system, the functional requirement is:

- Recommending counter measures that must be taken when an alert message is displayed

The domain expert carried out the functional testing and gave the aviation expert system his seal of approval.

6.3.2 Performance Testing

When the system performs the functions that meet the requirements, testing on the non-functional requirements are needed. During this stage, only volume testing is necessary. Volume testing addresses the handling of large amounts of data in the system. Because of the tremendous data involved in the aviation expert system, this test failed when more and more systems reached completion. A 'code array too small' error message was encountered.

The author managed to successfully debug the error by doing research on the size of the code array and rectified the problem by enlarging the default code size.

6.3.3 Acceptance Testing

When functionality and performance testing have been completed, the system is assumed to have met all requirements specified during the initial stages of the software development life cycle. Acceptance testing is then done to obtain the user's approval of the system's overall functionality.

6.3.4 Installation Testing

Installation testing is the final phase, which is carried out once the user has approved the overall functionality of the program. The aviation expert system is a stand-alone program that allows a user to install it on a personal computer. Hence, the test is done to ensure the program works on various platforms and functions accordingly. The aviation expert system was tested on Windows 95/98/2000/NT/XP and has been found to work perfectly on these platforms with no disruption to its functionality.

6.3.5 Additional Testing

Some techniques used for system testing are informal and are practised only by very experienced programmers. Incremental coding is one of these techniques and it is particularly suited for this

application. Incremental coding is a method whereby a small portion of the module is written and tested with simple data. Thereafter this portion is modified as necessary and the scope is widened to suit the requirements of the user. This is a repetitive process that occurs throughout the development of the application.

6.4 Conclusion

The aviation expert system has been tested and found to have met the requirements of the user. Various test phases were carried out to ensure functional and non-functional requirements were met. The following chapter evaluates the aviation expert system and discusses its strengths and limitations.

Chapter 7

System Evaluation

7.0 Introduction

The aviation expert system was proposed to be a stand-alone application developed for trainee pilots to be used in simulation cockpits. The system is able to advise trainee pilots on counter measures to take if there is an alert message(s) on the EICAS screen in the simulation cockpit and to give them faster access to the checklist.

Four of the 16 systems are included in the application only for knowledge purposes. They are Automatic Flight; Communications; Flight Instruments, Displays; and Warning Systems. The aviation expert system uses rule-based reasoning as its problem solving method.

Therefore, the strengths, limitations, problems and solutions of the system will be evaluated based on the above scope.

7.1 System Strengths

User Friendliness

The aviation expert system was built using Visual Prolog 5.2, which provides a graphical user interface that is similar to other Windows applications in the market. Consistency in screen resolution and control buttons maintains the discipline of the system. Therefore, functions like the 'Help' and '←' buttons can be found on almost every screen to enable the user to request for help or backtrack to the previous screen.

System Transparency

The aviation expert system employs system transparency, whereby users need not know much about rule-based reasoning, the problem solving methodology used to produce the advice. The user only works with the user interface that consists of controls such as push buttons and radio buttons.

Proper Documentation

A complete and well-defined documentation that covers every aspect from the planning stages to the development phase is provided. In order to facilitate maintenance and future enhancements, Visual Prolog has specially generated documentation that allows other programmers to locate and modify the relevant clauses with ease. In addition, a user manual is provided to assist new users in using the system.

Reliability

The aviation expert system is reliable. It behaves and functions just as stated in the systems specification. The system will not produce any dangerous or costly failures if it is used in the manner as stated in the systems manual.

Modularity

Modularity refers to features in the system design where the system is developed using a modular approach. The aviation expert system can be easily enhanced in the future; meaning changes in functionality or scope can be made without difficulty. The ability to do this comes from the program, which will be written in modular. This also allows reusability of certain functions and common procedures.

Consistency

Certain physical features such as screen appearances have a similar design to ensure consistency of the system. This portrays a disciplined and formal system that exhibits professionalism. Commands like 'Help' and '←' are displayed on every main screen to enable the user to request for help or backtrack to the previous screen. This also indirectly enhances the simplicity of the program to enable new users who are not proficient with computers to use the application with ease.

Simple Installation

The aviation expert system works on most Windows platforms (Win 95/98/2000/NT/XP). It can be run on any one of these machines directly from its executable file. Visual Prolog does not need to be installed on a computer for the aviation expert system to function. All you have to do is copy the 'Exe' folder generated by Visual Prolog to a computer and double-click on the Aviation Expert System executable file inside the 'Exe' folder.

7.2 System Limitations

Supports Only Bitmaps

Visual Prolog only supports bitmap images. Therefore all images had to be converted into bitmaps, which was done using Adobe Photoshop.

Limited User Interface Components

The user interface was designed using the Application Expert and the Dialog Package Expert, which utilises the 'drag and drop' feature, very much similar to that of Visual Basic and Visual C++. Unfortunately, unlike these tools, the interface components of Visual Prolog are limited to a set of standard controls making it impractical to create a modern and attractive interface.

7.3 Future Enhancements

Enlarge Current Scope

More simulation tests can be included in the Multiple Failure section to better prepare the trainee pilots to cope with non-normal situations.

7.4 Problems and Solutions

This section discusses in brief, the problems faced by the author and the measures undertaken to overcome them.

Unfamiliarity With Development Tool

Although the author had completed a course in Prolog (WAES2201), it was still an uphill task to master Visual Prolog 5.2, as the course undertaken was more theoretical than practical. The author used Visual Prolog manuals and the Internet as resources to study this tool. Besides that, the author also acquired help from associates who were proficient with the language.

Inability To Link Interface With Coding

The Visual Prolog manuals only contained details of the functions and sets of codes that could be used to obtain certain relevant features. The limited examples given were hard to comprehend. Furthermore, it only explained what could be used to obtain the feature, and not how. This problem was overcome by acquiring help from associates who were proficient with the language and posting questions on the Visual Prolog Discussion Forum.

Inability To Insert Images

The Visual Prolog manuals provided a theoretical explanation about codes that were needed to insert images but did not mention how to implement these codes. The author posted this inquiry on the Visual Prolog discussion forum and received tremendous feedback on how to insert an image into a dialog.

Code Array Too Small

The default code array set by Visual Prolog was too small for the ever-growing lines of code. The author conducted some research on how the code array could be expanded and attained a

solution that was able to rectify the problem. The solution involved the modifying of codes in the VPI Tools.pro file, which would resize the code array to 64,000 compared to the default size of 5,000.

```
ifdef platform_16bit
  code = 5000
elsedef
  code = 64000 %set your code size > 32000 if have "Code array too small" problem
enddef
```

7.5 Conclusion

The aviation expert system has managed to achieve its objectives and overcome constraints that plagued its development. The system covers every aspect the author proposed in WXES 3181.

Chapter 8

Conclusion

8.0 Introduction

This is the final chapter of this book. It concludes the whole final year project with a general overview of the aviation expert system and the knowledge and experience gained by the author throughout the development of this system.

8.1 General Overview

The seven chapters in this book sum up the software development life cycle of the aviation expert system. Chapter one is an introduction to the system that covers the objective, scope, motivation and project milestone. The following chapter is the literature review conducted for the aviation expert system. It includes research on expert systems and aviation, and a study of a system that has similar characteristics to the aviation expert system.

The third chapter describes the methodology undertaken to develop the aviation expert system. It gives a detailed analysis on rule-based reasoning and rule-based expert systems. The fourth chapter discusses the requirements analysis, such as software and hardware requirements, and system design, which includes the system design architecture and graphical user interface. The following chapters describe the system implementation, system testing and system evaluation. These chapters explain how the aviation expert system was developed, the problems faced during the development life cycle and the measures taken to overcome them, and the numerous tests performed to make sure the system was error-free.

8.2 Knowledge and Experience Gained

Throughout the undertaking of this final year project, the author has gained immense knowledge and experience, which is summarized in the following section.

8.2.1 Knowledge Gained

The author is certainly more knowledgeable about aviation than he ever was. The author is able to identify major parts of an airplane and relate their functions. The author is also familiar with the mechanics of flight and the many complex systems of an airplane.

The author's familiarity with Artificial Intelligence because of the courses undertaken as part of the Computer Science Degree Program, was deepened through this final year project as the author had to apply what he knew to develop the aviation expert system. The author can now better comprehend rules, reasoning, expert systems and the Prolog programming language.

8.2.2 Experienced Gained

The experience gained throughout the development of the aviation expert system is priceless and will benefit the author in many ways. Developing an application and understanding as well as implementing each phase of the Software Development Life Cycle has given the author the confidence to venture into the working world.

8.3 Conclusion

The expert system was developed using Visual Prolog while implementing rule-based reasoning as the problem solving approach. The objective of the aviation expert system is to advise trainee pilots on the course of action to take when dealing with a failure situation in simulation cockpits. In the wake of the September 11th terrorist attacks against the United States of America, it was the author's desire to include a separate module that dealt with such elements, but was informed by the domain expert that the information is deemed classified.

The author implemented every phase of the Software Development Life Cycle in developing the aviation expert system. After the coding of the system, system testing was carried out to identify flaws and bugs in the system. Finally the system was evaluated from all perspectives. A user manual, which assists users in handling the system in the approved manner, was created. The aviation expert system has been completed and is now ready for use.

The knowledge and experience the author gained throughout this final year project is priceless and shall be put to good use in the future.

Reference

Reference

[1] Boeing Company
http://www.boeing.com/commercial/safety/statistics.htm
Date Referred: 4.4.2002

[2] *Quick Reference Handbook*, Boeing Operations Manual, The Boeing Company.

[3] Durkin, John (1994). *Expert Systems Design and Development*, Maxwell Macmillan.

[4] Literature Review
http://www.utoronto.ca/hswriting/lit-review.htm
Date Referred: 11.4.2002

[5] Aeronautics Learning Laboratory for Science Technology, and Research (ALLSTAR)
http://www.allstar.fiu.edu
Date Referred: 4.4.02

[6] Science Direct
http://www.sciencedirect.com/
Date Referred: 5.4.02

[7] *Getting Started*, Visual Prolog 5.0 Manual, Prolog Development Corporation.

[8] LPA WIN-PROLOG
http://www.lpa.co.uk/win.htm
Date Referred: 15.4.2002

[9] Pfleeger, S.L (1998). *Software Engineering*, Prentice-Hall International, pg. 135-248.

[10] LPA flex
http://www.lpa.co.uk/flx.htm
Date Referred: 25.4.2002

[11] Kendal & Kendal. *Systems Analysis and Design*, Fourth Edition, Prentice Hall.

Appendixes

Appendix A

User Manual

Abstract

The aviation expert system is a stand-alone application developed for trainee pilots to be used in simulation cockpits. It advises trainee pilots on counter measures to take if there is an alert message on the EICAS screen in the simulation cockpit.

The objective of this user manual is to provide guidance to the system administrator and the targeted users on installing and using the aviation expert system.

Table of Contents

List of Figures

A.1: SYSTEM OVERVIEW

A.1.0Introduction

The aviation expert system is a stand-alone application developed to assist trainee pilots in simulation cockpits. It advises them on counter measures to take if there is an alert message on the EICAS screen in the simulation cockpit. The aviation expert system also prepares trainee pilots for multiple failure situations using simulation tests.

This user manual is meant to guide the system administrator and the targeted users in installing and using the aviation expert system in the approved manner. Since the system is a stand-alone application, its prerequisites are minimal.

A.1.1System Objectives and Functionality

The aviation expert system is able to fulfill these objectives:

- Advise trainee pilots on counter measures to take if there is an alert message on the EICAS screen in the simulation cockpit.
- Advise trainee pilots on counter measures to take if there are multiple alert messages on the EICAS screen in the simulation cockpit through simulation tests.
- Give trainee pilots faster and easier access to the checklists.

A.1.2System Features

A.1.2.1User Friendliness

The aviation expert system was built using Visual Prolog 5.2, which provides a graphical user interface that is similar to other Windows applications in the market. Consistency in screen resolution and control buttons maintains the discipline of the system. Therefore, functions like

the 'Help' and '←' buttons can be found on almost every screen to enable the user to request for help or backtrack to the previous screen.

A.1.2.2 Help Files

There are help buttons on every screen to assist users in carrying out the right course of action and using the system in the approved manner. Clicking the help button for a particular screen will invoke a dialog that explains in detail what the user must do at that screen.

A.1.2.3Reliability

The aviation expert system is reliable. It behaves and functions just as stated in the systems specification. The system will not produce any dangerous or costly failures if it is used in the manner as stated in the systems manual.

A.1.2.4Consistency

Certain physical features such as screen appearances have a similar design to ensure consistency of the system. This portrays a disciplined and formal system that exhibits professionalism. Commands like 'Help' and '←' are displayed on every main screen to enable the user to request for help or backtrack to the previous screen. This also indirectly enhances the simplicity of the program to enable new users who are not proficient with computers to use the application with ease.

A.1.2.5Simple Installation

The aviation expert system works on most Windows platforms (Win 95/98/2000/NT/XP). It can be run on any one of these machines directly from its executable file. Visual Prolog does not need to be installed on a computer for the aviation expert system to function. All you have to do is copy the 'Exe' folder generated by Visual Prolog to a computer and double-click on the Aviation Expert System executable file inside the 'Exe' folder.

A.1.2.6System Transparency

The aviation expert system employs system transparency, whereby users need not know much about rule-based reasoning, the problem solving methodology used to produce the advice. The user only works with the user interface that consists of controls such as push buttons and radio buttons.

A.1.3Copyright

A.1.4Conclusion

The aviation expert system has achieved its objective of assisting trainee pilots in simulation cockpits. Future enhancements include more simulation tests can be included in the Multiple Failure section to better prepare the trainee pilots to cope with non-normal situations.

A.2 INSTALLATION GUIDE

A.2.0Hardware Requirements

Minimum hardware requirements:

- 32MB RAM
- 2.1 GB of hard disk space
- Pentium Processor at 133 MHz
- 512KB Pipeline Burst Cache
- 4MB Graphic Card
- Keyboard
- Mouse

A.2.1Software Requirements

Minimum software requirements:

- Widows 95/98/2000/XP

Since the aviation expert system is a stand-alone application, it doesn't require any additional software to function, including Visual Prolog 5.2. The aviation expert system was tested on all Windows platforms and has been found to work perfectly with no disruption to its functionality.

A.2.2 Installation Guide

Step-by-step installation guide:

1) Copy the EXE folder to a selected drive on a PC preinstalled with Windows
2) Double-click on the Aviation Expert System.exe file in the EXE folder to start the application
3) Place a shortcut on the desktop for easy access

A.3 USERS GUIDE

A.3.0Introduction

The aviation expert system assists trainee pilots in dealing with two failure situations:

- Single
- Multiple

A.3.1Single

The aviation expert system advises trainee pilots on counter measures to take if there is an alert message on the EICAS screen.

List of screens:

1) Main Menu
2) Failure
3) Systems
4) System
5) Condition
6) Counter Measures
7) Reason
8) Meaning
9) Help

A.3.1.1Main Menu

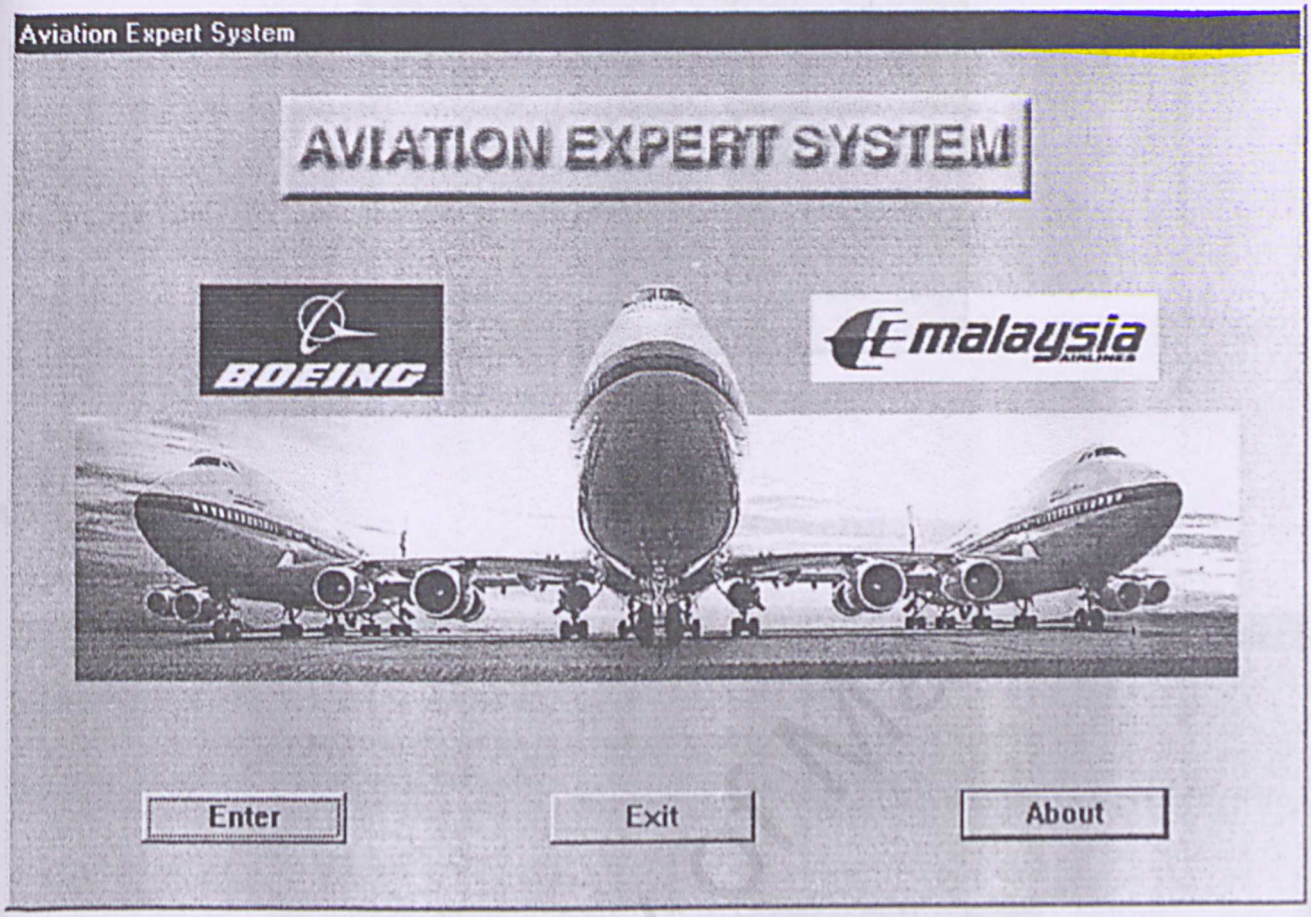


Figure A.3.1 Main Menu

The Main Menu has three functionalities:

- Enter – Invokes the Failure screen
- Exit – Terminates the application
- About – Invokes the About screen

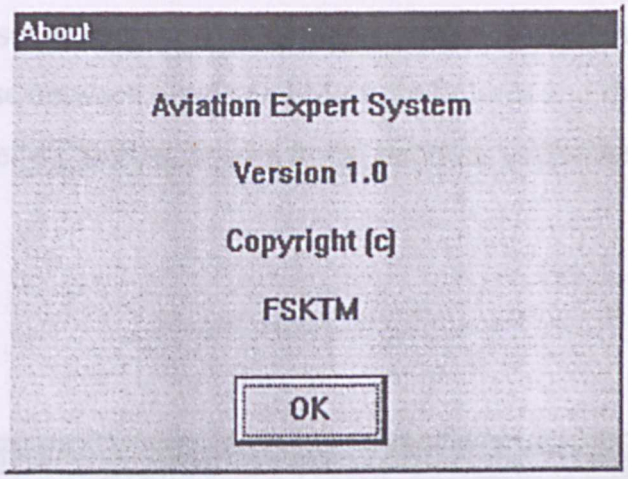


Figure A.3.2 About Screen

A.3.1.2Failure

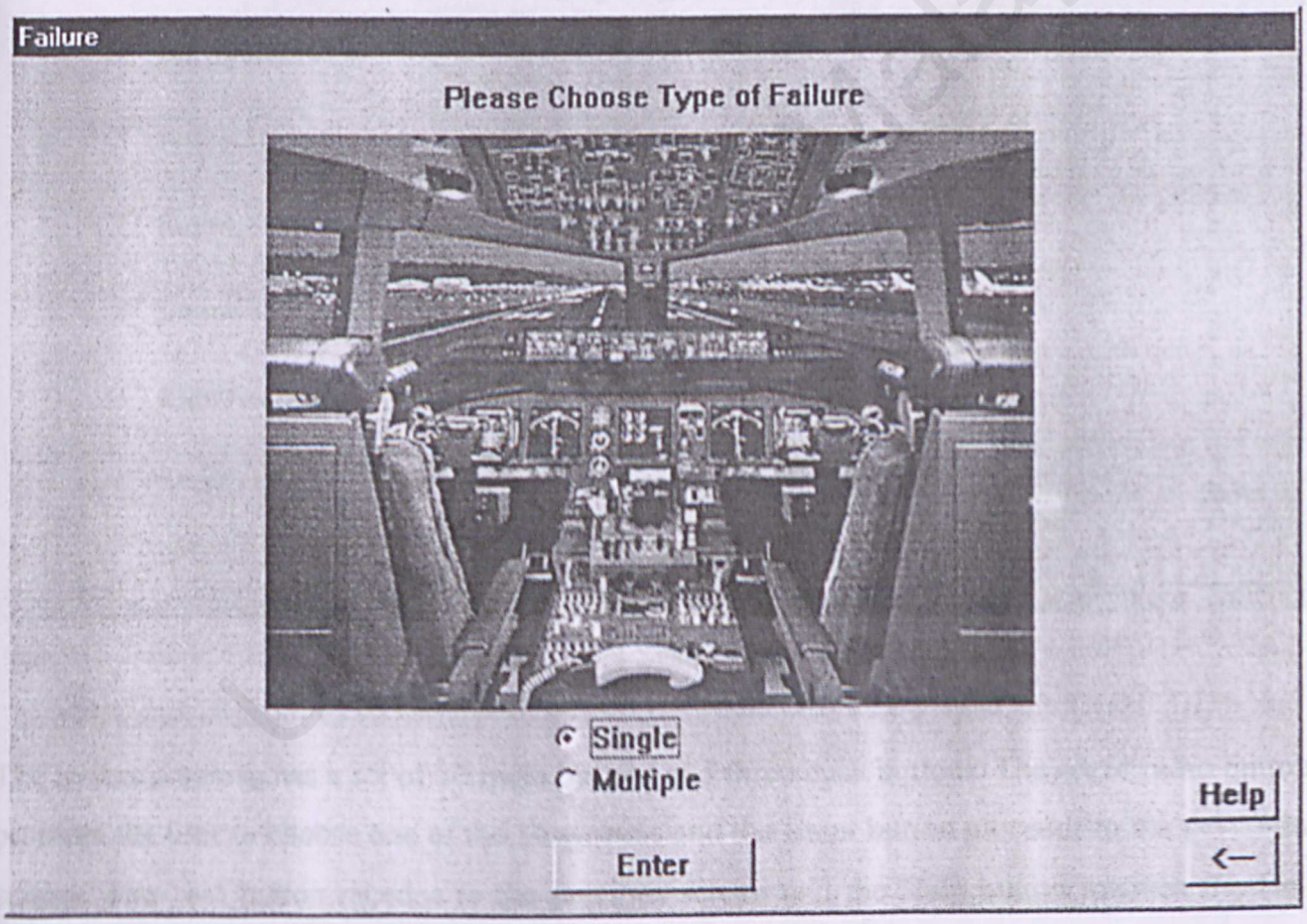


Figure A.3.3 Failure Screen

The Failure screen has a set of two radio buttons and three push buttons. The set of radio buttons prompts the user to choose between Single and Multiple failures and the Enter button proceeds to the pertaining screen. The '←' button recedes to the previous screen and the Help button invokes the Help screen.

A.3.1.3 Systems

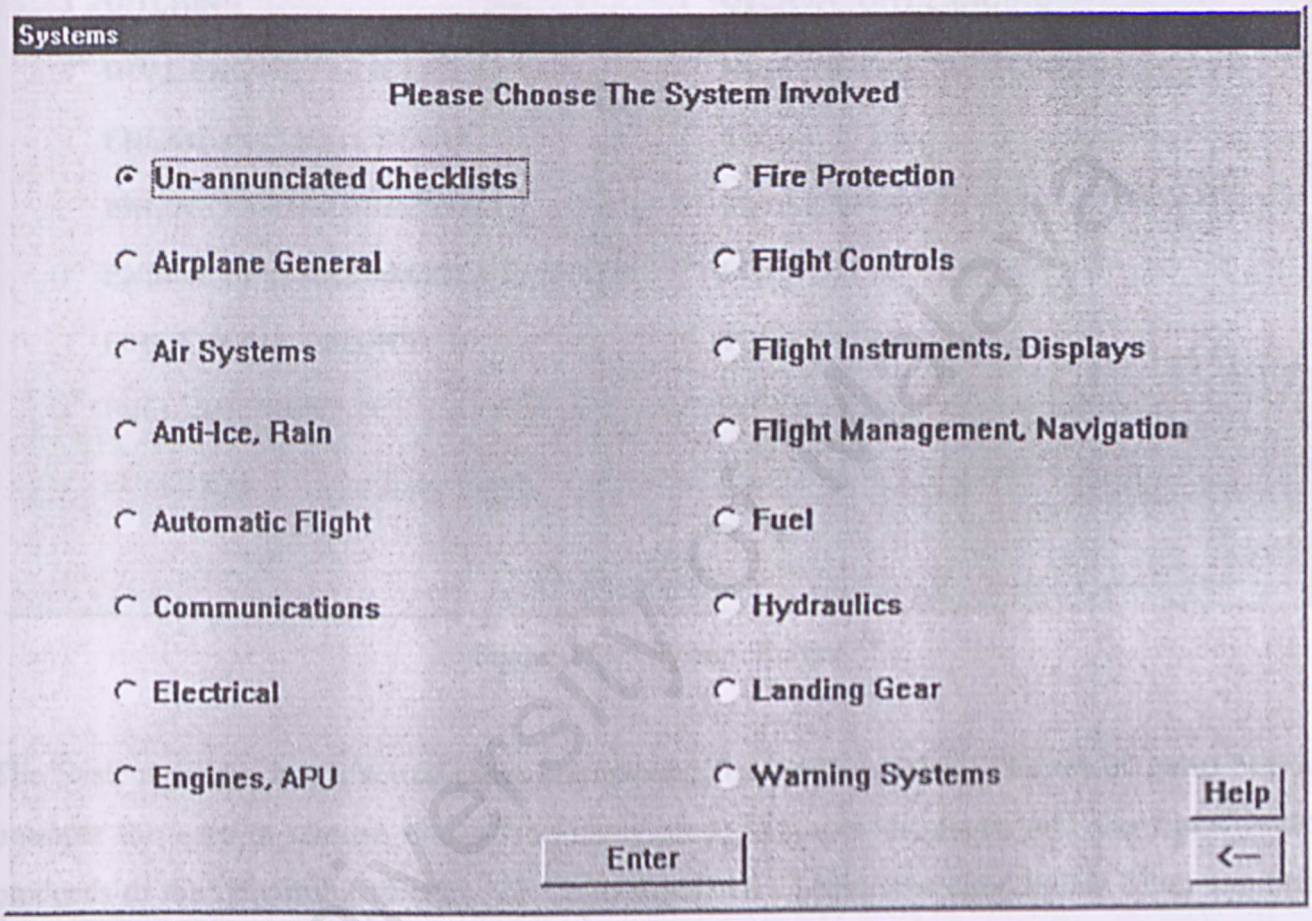


Figure A.3.4 Systems Screen

The Systems screen has a set of 16 radio buttons and three push buttons. The set of radio buttons prompts the user to choose one of the 16 systems and the Enter button proceeds to the pertaining screen. The '←' button recedes to the previous screen and the Help button invokes the Help screen.

A.3.1.4System

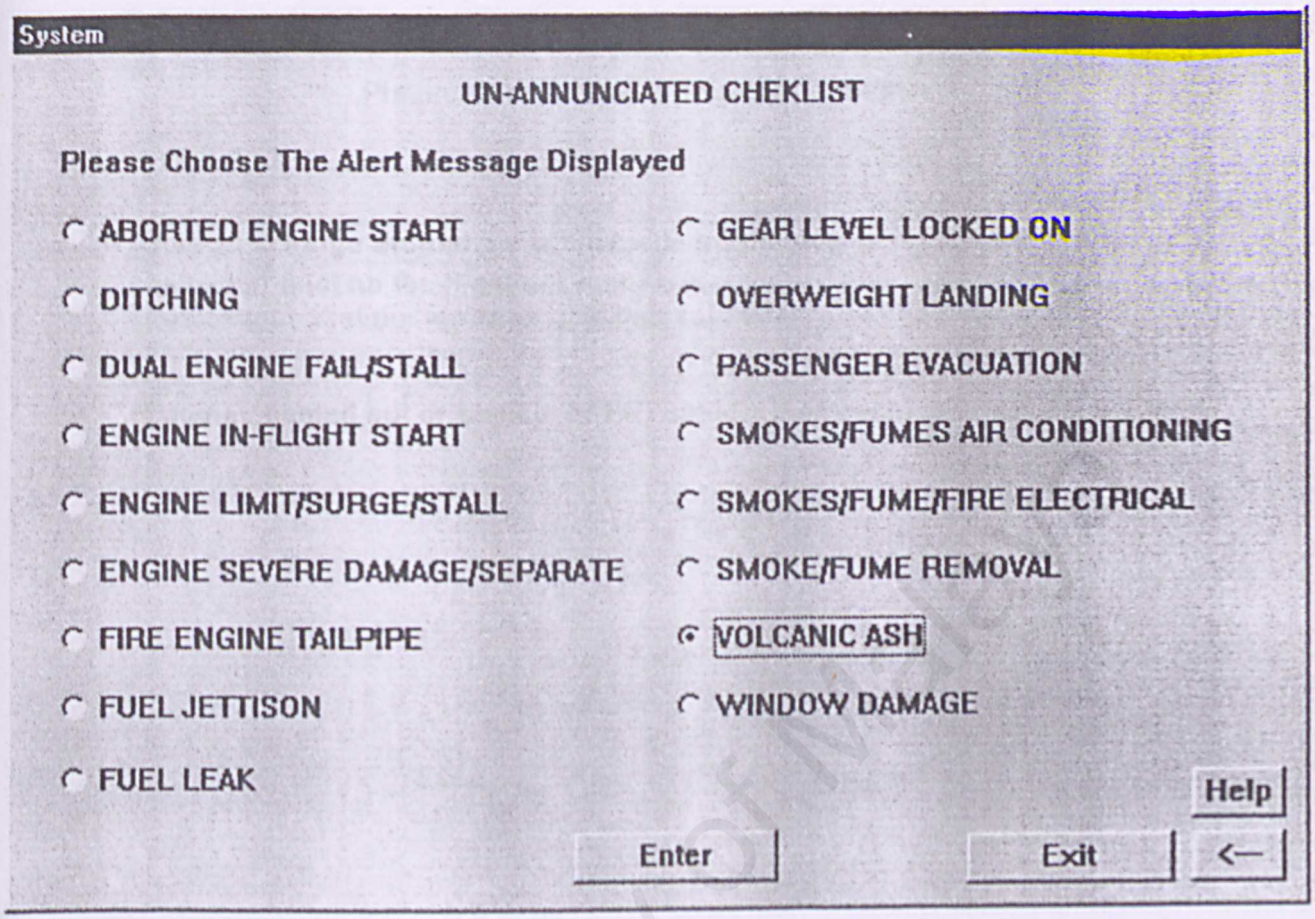


Figure A.3.5 System Screen

The System screen has a set of radio buttons and four push buttons. The set of radio buttons prompts the user to choose one of the many alert messages displayed and the Enter button proceeds to the pertaining screen. The Un-annunciated Checklist is cited below as an example. The '←' button recedes to the previous screen, the Help button invokes the Help screen and the Exit button terminates the application.

A.3.1.5Condition

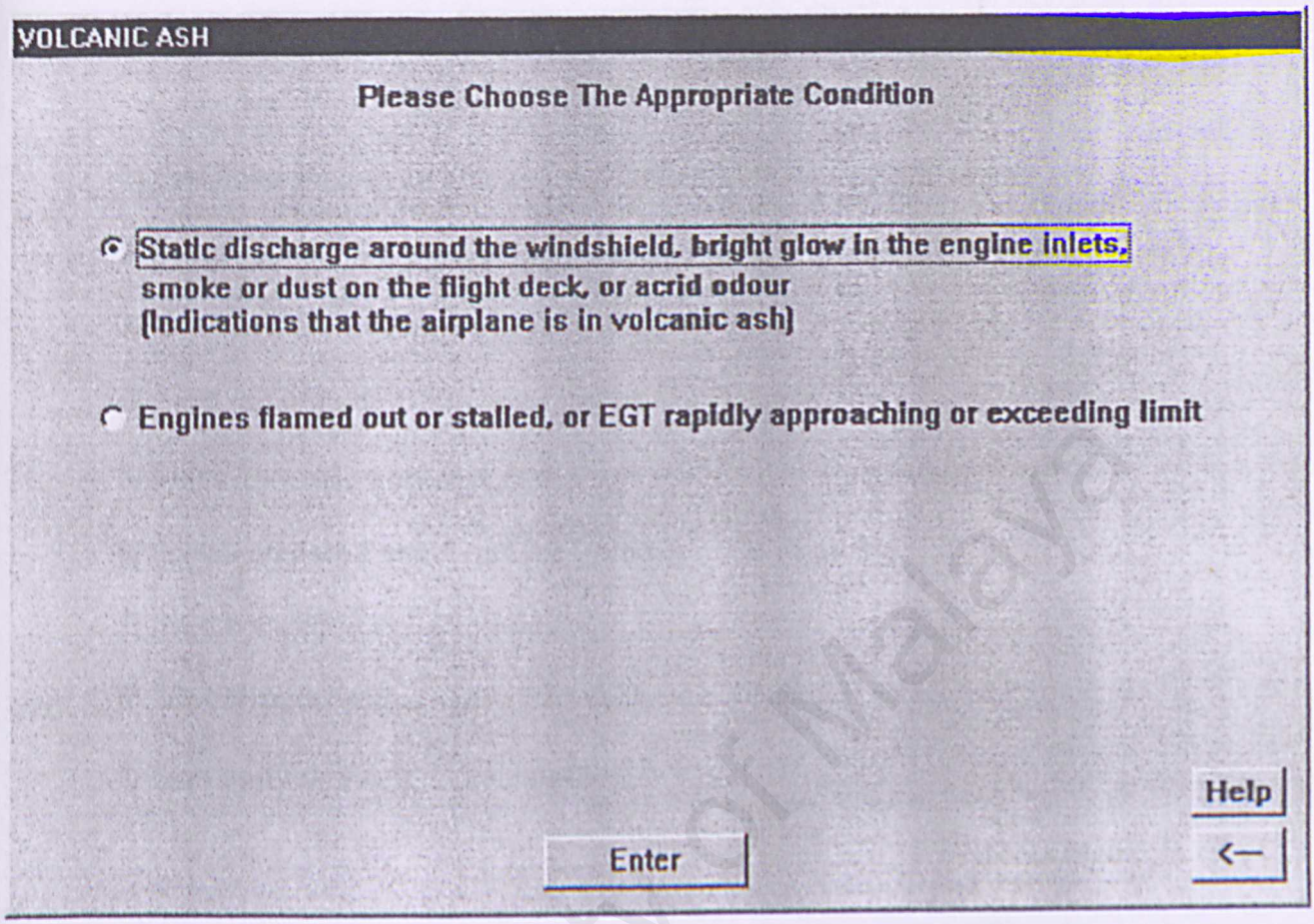


Figure A.3.6 Condition Screen

The Condition screen has a set of radio buttons and three push buttons. The set of radio buttons prompts the user to choose one of the many conditions encountered and the Enter button proceeds to the pertaining screen. The VOLCANIC ASH alert message of the Un-annunciated Checklist is cited as an example. The '←' button recedes to the previous screen and the Help button invokes the Help screen.

A.3.1.6 Counter Measures

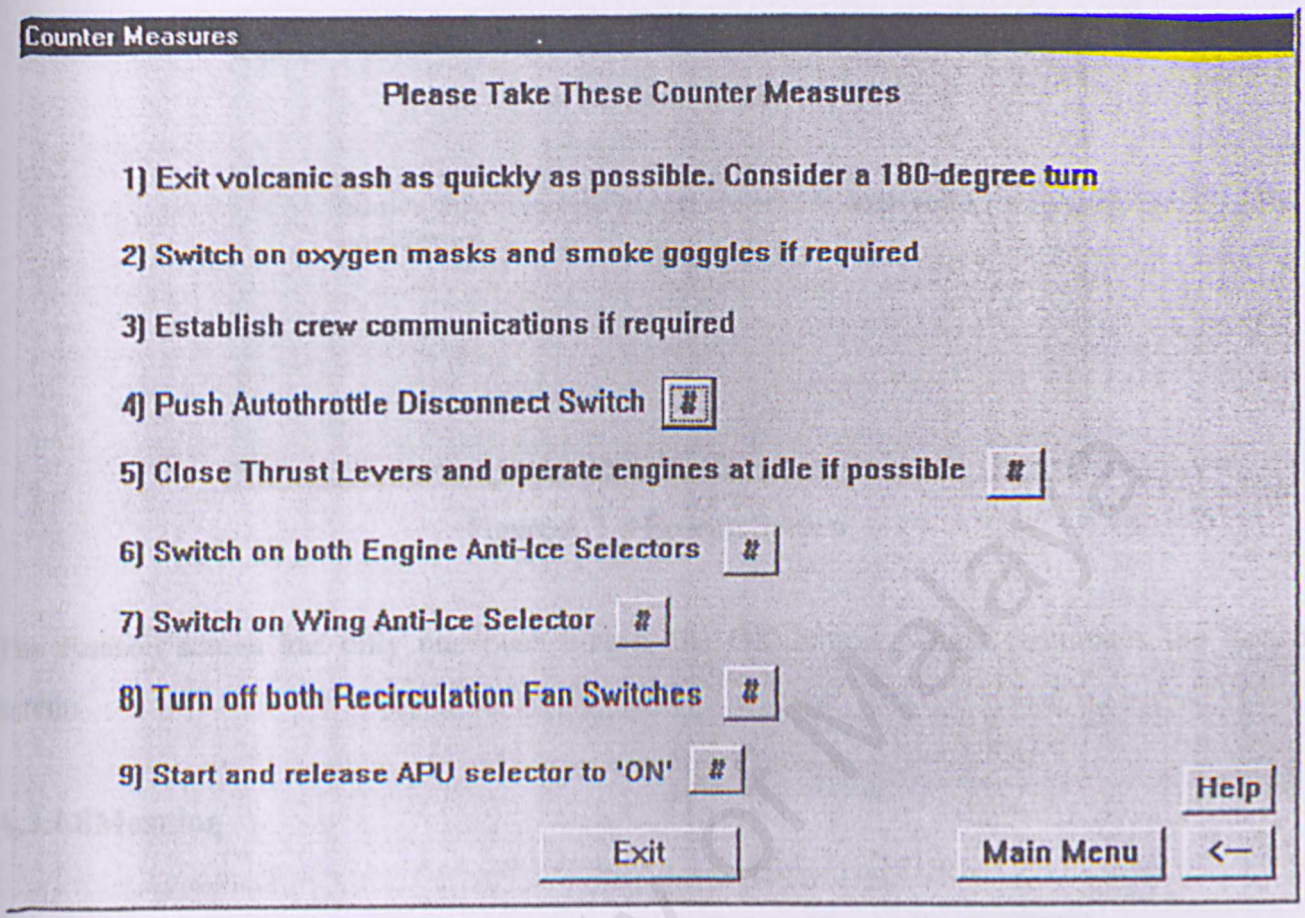


Figure A.3.7 Counter Measures Screen

The Counter Measures screen has five functionalities:

- '#' – Invokes the Reason screen for a particular counter measure
- Exit – Terminates the application
- Main Menu – The application returns to the Main Menu
- '←' – The application recedes to the previous screen
- Help – Invokes the Help screen

A.3.1.7Reason

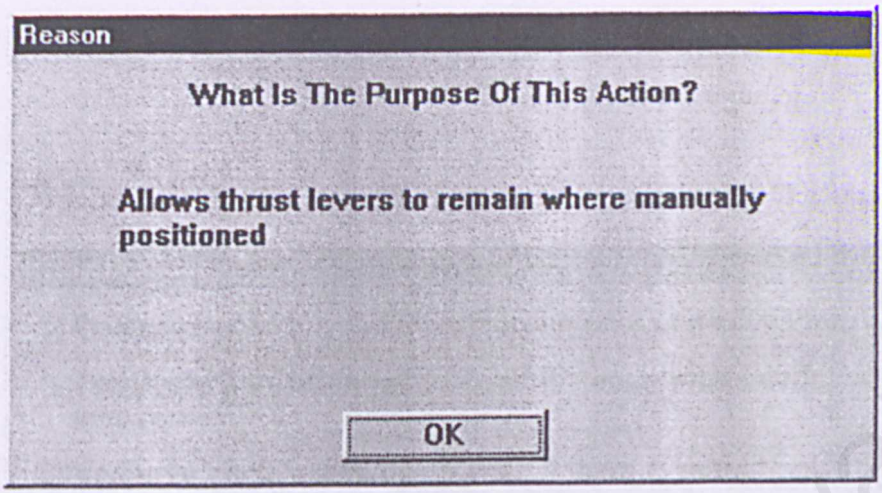


Figure A.3.8 Reason Screen

The Reason screen has only one push button, the OK button, which terminates the Reason screen.

A.3.1.8Meaning

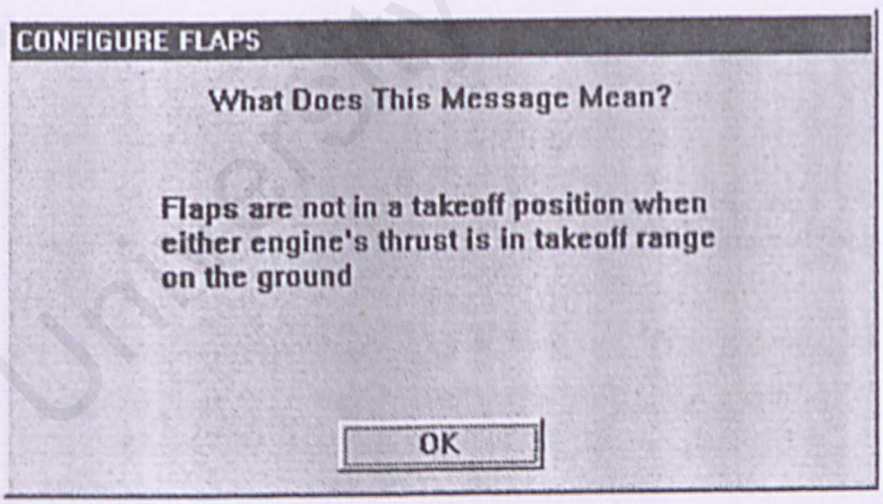


Figure A.3.9 Meaning Screen

Sometimes, the System screen does not proceed to the Condition screen to prompt the user for the conditions encountered for the alert message selected. This is because the alert messages displayed by these systems are solely for information purposes and do not require counter

measures. Therefore, the Meaning screen just explains to the user the why the alert message was displayed. The CONFIGURE FLAPS alert message of the Warning Systems is cited as an example.

A.3.1.9Help

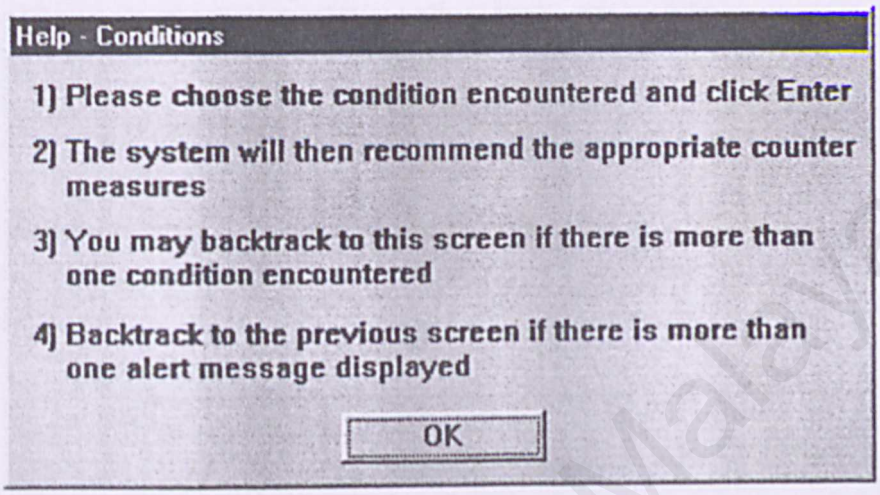


Figure A.3.10 Help Screen

The Help screen has only one push button, the OK button, which terminates the Help screen. The Help screen for the Condition screen is cited as an example.

A.3.2 Multiple

The aviation expert system also assists trainee pilots in dealing with multiple failure situations through simulation tests.

List of screens:

1) Test
2) Test X
3) Solutions
4) Help

A.3.2.1 Test

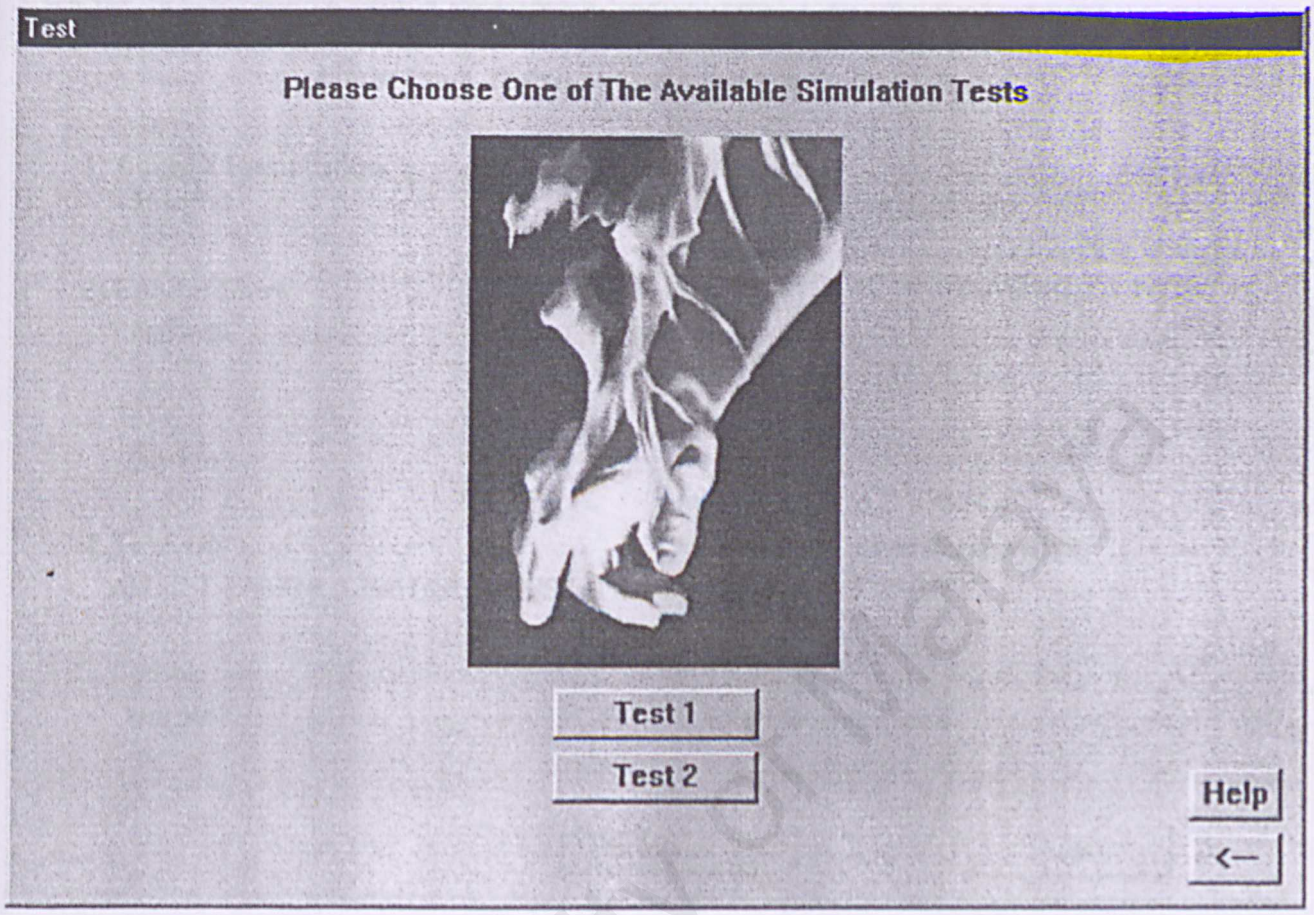


Figure A.3.11 Test Screen

The Test screen has four functionalities:

- Test 1 – Invokes the Test 1 screen
- Test 2 – Invokes the Test 2 screen
- '←' – The application recedes to the previous screen
- Help – Invokes the Help screen

A.3.2.2Test X

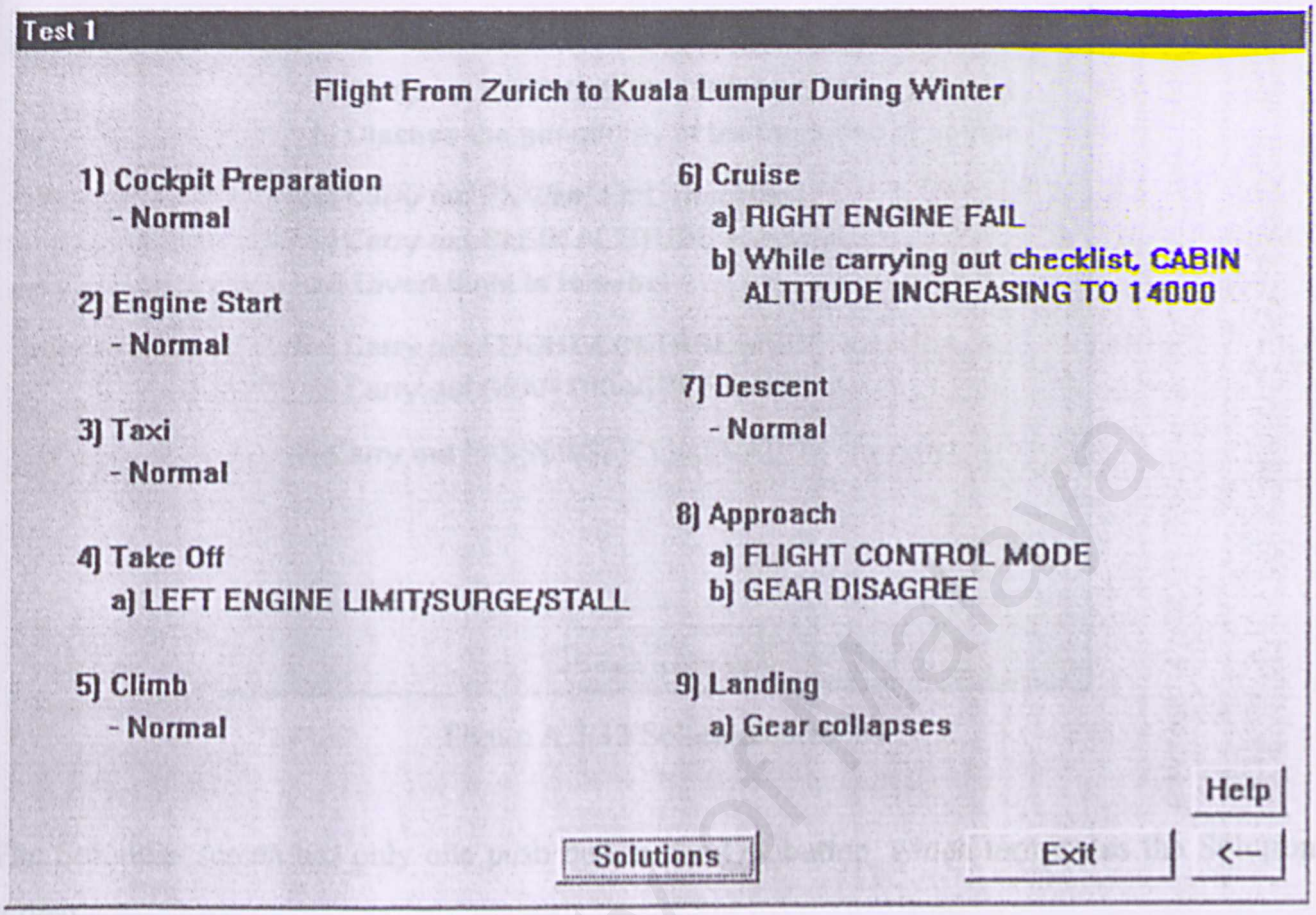


Figure A.3.12 Test 1 Screen

The Test X screen has four functionalities:

- Solutions – Invokes the Solutions screen
- Exit – Terminates the application
- '←' – The application recedes to the previous screen
- Help – Invokes the Help screen

A.3.2.3Solutions

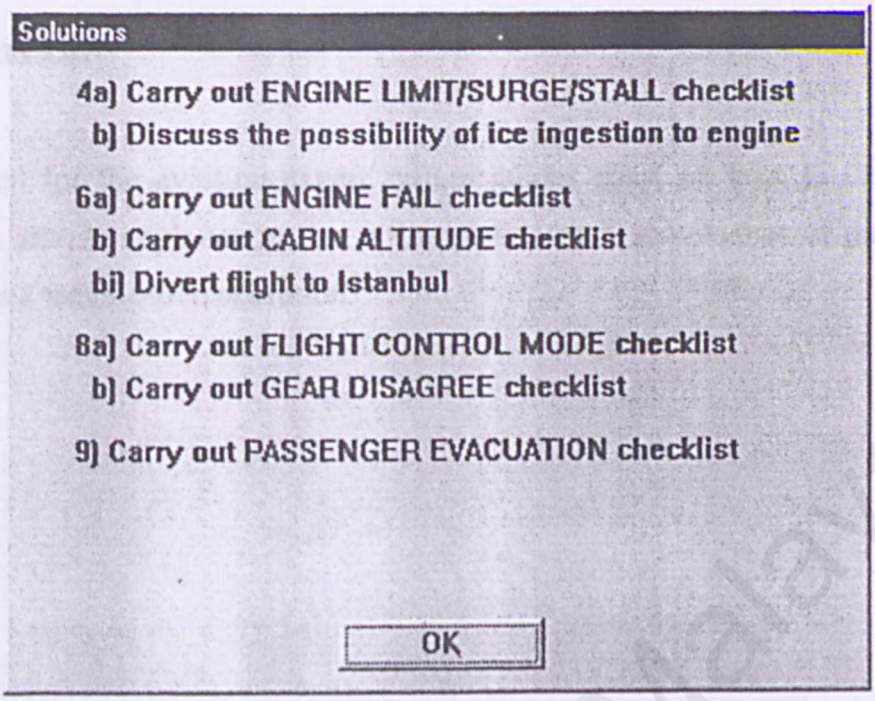


Figure A.3.13 Solutions Screen

The Solutions screen has only one push button, the OK button, which terminates the Solutions screen.

A.3.2.4Help

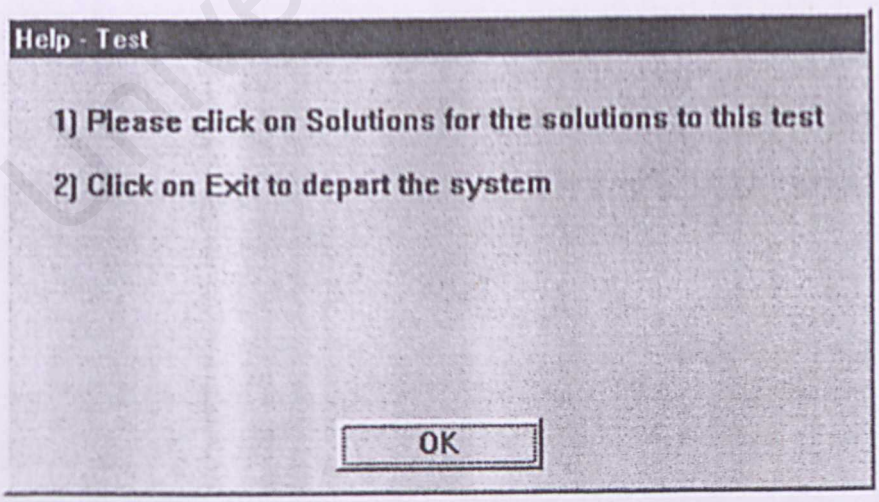


Figure A.3.14 Help Screen

A.4 CONCLUSION

A.4.0Conclusion

The User Manual for the aviation expert system guides users on how to install and use the application. The simple explanation that accompanies each screen shot of the aviation expert system facilitates a trouble-free operation.

Appendix B

Interview Questions

Interview Questions

1. What systems are there in an airplane?
2. How do pilots know what counter measures to take when there is a failure during flight?
3. What are checklists? (Based on answer of previous question)
4. What are the disadvantages of the current system?
5. Can you provide me with the list of alert messages and the corresponding counter measures?
6. Can you provide me with examples of simulation tests for multiple failure situations you have to undergo every now and then?
7. Can you provide me with a list of counter measures that the flight crew employs when faced with threats from terrorists or hijackers?
8. Would you support the idea to implement this expert system if it can overcome the disadvantages of the current system?
9. What suggestions do you have for this expert system?
10. Would you prefer an electronic checklist to a manual one?

Appendix C

Sample Coding

Sample Coding

This section comprises of the main sets of coding that were utilised in the development of the aviation expert system. For documentation purposes, the codes have been shortened for simplicity. The sets of coding were used in:

- Creating dialogs
- Destroying dialogs
- Handling radio buttons
- Inserting images

Creating Dialogs

```
%BEGIN Main, idc_about _CtlInfo
 dlg_main_eh(_Win,e_Control(idc_about,_CtrlType,_CtrlWin,_CtlInfo),0):-!,
        dlg_about_dialog_Create(_Win),      % creates the 'About dialog' screen
        !.
%END Main, idc_about _CtlInfo
```

When the 'About' button on the 'Main' screen is clicked, this clause is activated and the 'About dialog' screen is invoked.

Destroying Dialogs

```
%BEGIN Main, idc_exit _CtlInfo
 dlg_main_eh(_Win,e_Control(idc_exit,_CtrlType,_CtrlWin,_CtlInfo),0):-!,
        win_Destroy(_Win),                  % destroys the 'Main' screen
        !.
%END Main, idc_exit _CtlInfo
```

When the 'Exit' button on the 'Main' screen is clicked, this clause is activated and the 'Main' screen is destroyed.

The codes for creating and destroying a dialog can also be used together as shown below:

```
%BEGIN Main, idc_enter _CtlInfo
 dlg_main_eh(_Win,e_Control(idc_enter,_CtrlType,_CtrlWin,_CtlInfo),0):-!,
        win_Destroy(_Win),                  % destroys the 'Main' screen
        dlg_failure_Create(_Win),           % creates the 'Failure' screen
        !.
%END Main, idc_enter _CtlInfo
```

When the 'Enter' button on the 'Main' screen is clicked, this clause is activated and the 'Main' screen is destroyed while the 'Failure' screen is invoked.

Handling Radio Buttons

Before a radio button is used, a temporary 'box' is needed to store the state of the radio button. This is done by declaring a temporary storage in the *.pre file:

```
FACTS - box
 type (symbol)
```

Once this is done, the following code is used to assert the state of the radio button when it is selected. In a radio button group, only one button can be selected at a time. Therefore, the temporary 'box' is used to store the state of a specific button and retract it when another button is selected. The following codes do this:

```
%BEGIN Failure, idc_single _CtlInfo
 dlg_failure_eh(_Win,e_Control(idc_single,_CtrlType,_CtrlWin,_CtlInfo),0):-!,
        retractall(_,box),                  % retracts the contents of 'box'
        assert(type("s"),box),              % asserts type("s") into 'box'
        !.
%END Failure, idc_single _CtlInfo
```

```
%BEGIN Failure, idc_multiple _CtlInfo
 dlg_failure_eh(_Win,e_Control(idc_multiple,_CtrlType,_CtrlWin,_CtlInfo),0):-!,
      retractall(_,box),                    % retracts the contents of 'box'
      assert(type("m"),box),                % asserts type("m") into 'box'
      !.
%END Failure, idc_multiple _CtlInfo
```

The above coding is for the selection of type of failures. Selecting the 'single' option will assert the state type("s") while the 'multiple' option will assert the state type("m"). The following coding is for when the 'Enter' button is clicked after making a selection from the radio button group.

```
%BEGIN Failure, idc_enter _CtlInfo
   dlg_failure_eh(_Win,e_Control(idc_enter,_CtrlType,_CtrlWin,_CtlInfo),0):-
        type(X),
        X = "s",
        win_Destroy(_Win),                  % destroys the 'Failure' screen
        dlg_system_single_Create(_Win),     % creates the 'System_Single' screen
        !.

   dlg_failure_eh(_Win,e_Control(idc_enter,_CtrlType,_CtrlWin,_CtlInfo),0):-
        type(X),
        X = "m",
        win_Destroy(_Win),                  % destroys the 'Failure' screen
        dlg_system_multiple_Create(_Win), % creates the 'System_Multiple' screen
        !.
%END Failure, idc_enter _CtlInfo
```

Inserting Images

Visual Prolog does not have a specific feature for inserting images. However, the Custom Control tool of the Window and Dialog Editor can be programmed to display images. These are the steps that must be taken for an image to be displayed on a screen.

1) Save bitmap image in the Res\Win folder generated by Visual Prolog.
2) Add image to bitmap collection of current project using the Project Window.
3) Create a Custom Control in a particular dialog.
4) Declare the predicate and class for the Custom Control.

This is the sample coding for declaring the predicate and class and inserting the image:

```
class_Create("idc_main",e_hand),        % class declaration for the file main.bmp with class
                                        % name idc_main

e_hand : EHANDLER                       % predicate declaration

e_hand(W,e_update(_),0):-               % clause for inserting image
        Picture = pict_GetFromRes(idb_main),
        pict_Draw(W, Picture, pnt(0,0),12),
        pict_Destroy(Picture).
```